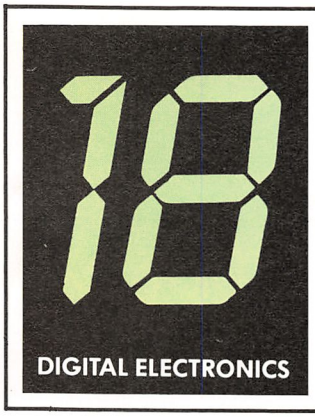


More about memories

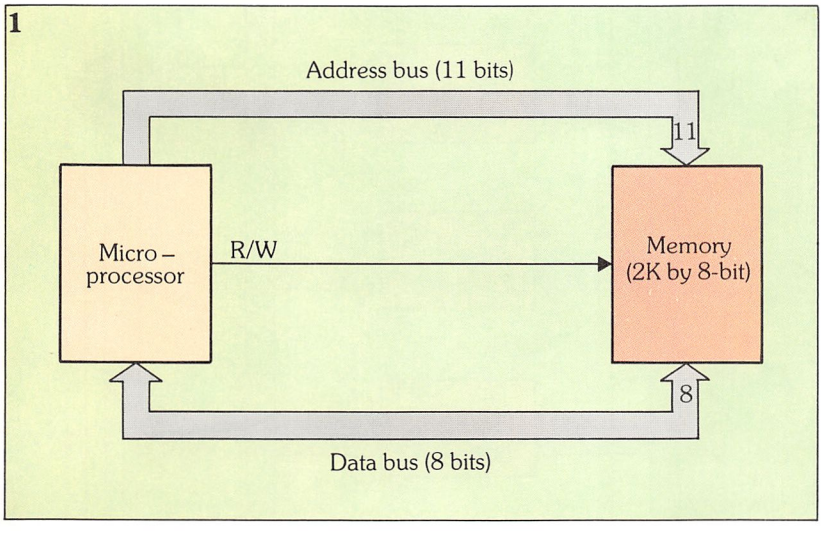
Making larger memories

A digital system may often have a requirement for a larger memory than is available on one component. The central processing unit of a computer, for example, may be capable of generating a 16-bit address, and therefore might address and use a total of 64K (2^{16}) by 16-bit memory locations. At the time of writing, the most cost effective semiconductor memory is a 64K by 1-bit RAM device; however, the situation is continually changing and devices are generally becoming cheaper as packing density and sales volumes increase. Obviously, manufacturers of digital systems take advantage of price differentials between memory devices, and design their digital system accordingly.

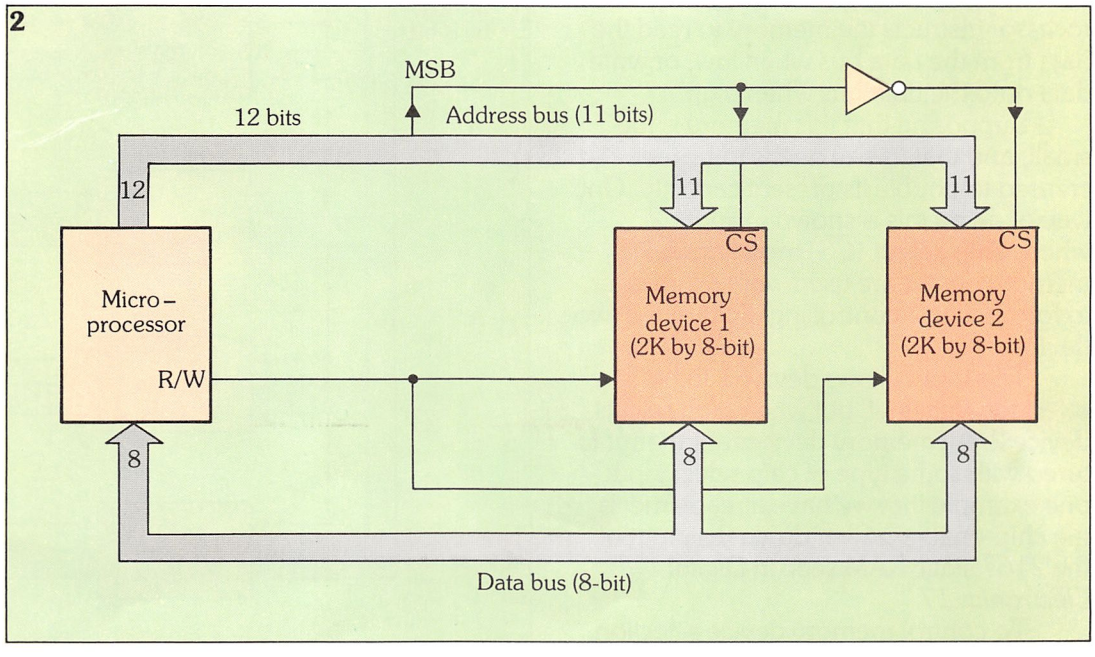
There is a large selection of RAM devices available for manufacturers to choose from: for small size, i.e. 32 by 1-bit, to very large, i.e. 256K by 1-bit. Inbetween these two limits there are many others, e.g. 256 by 8-bit, 8K by 1-bit, 8K by 8-bit. If a larger memory than a selected device is required, some means of joining devices must be possible. Also, each separate memory cell within each device must be uniquely addressable and all devices must be capable of connection onto a single data bus.

Connection onto a single data bus is

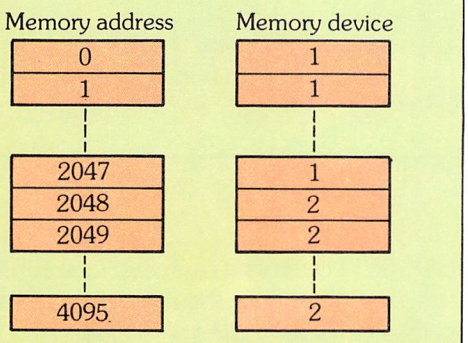
1. Microprocessor and memory connected by an address bus and data bus.



2. Doubling the memory capacity by using $\overline{CS}$ inputs to each device, with an inverter, forming a single controlling input.



3



no real problem. We have seen how the use of three-state or open collector output stages can provide this facility. But joining devices and unique addressing depends on the devices used, and on the size of the required memory. There are three basic ways devices may be joined and we'll look at these now.

Larger address

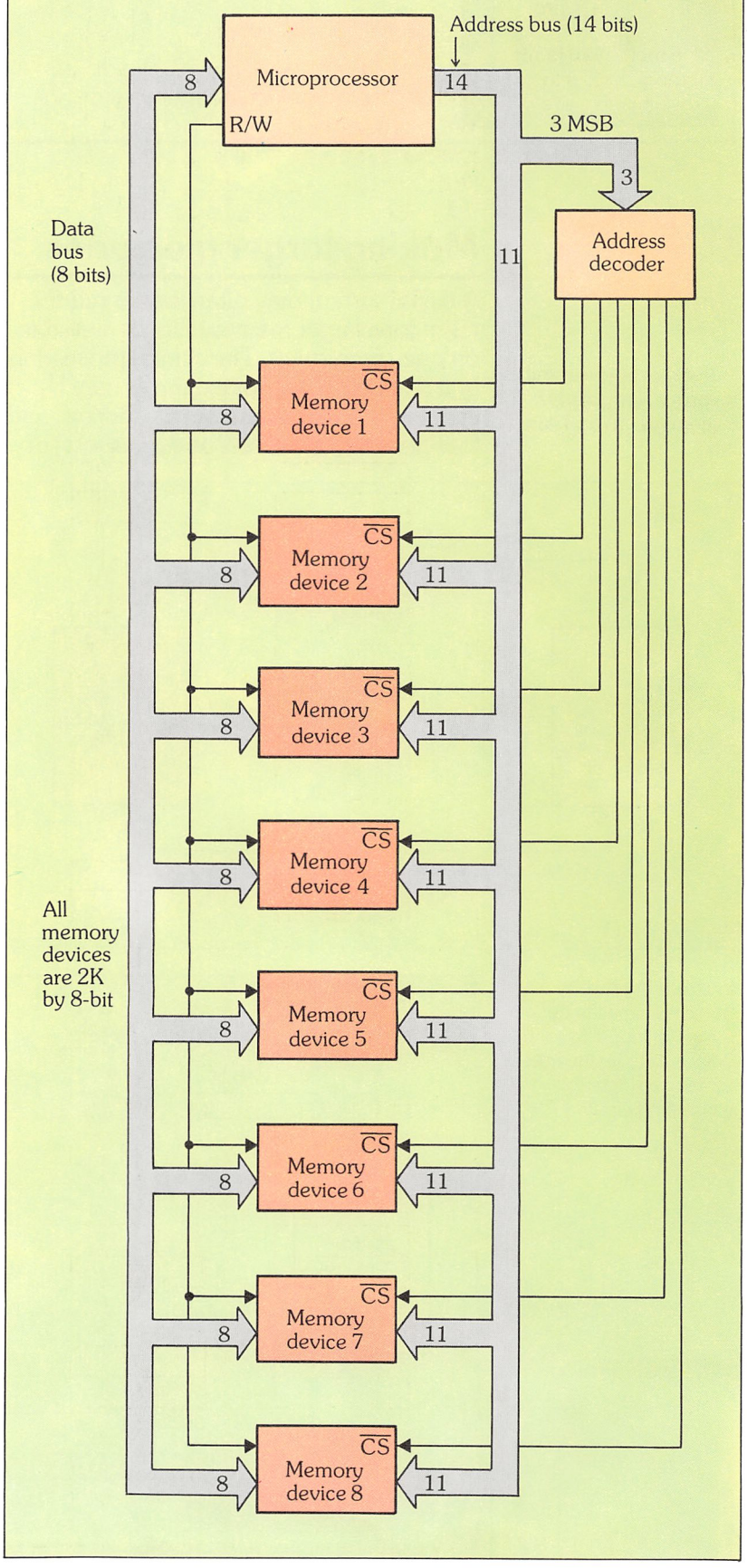
Figure 1 shows an example of a digital system formed by a microprocessor and a memory connected by the address bus and the data bus. The memory consists of a single 2K by 8-bit RAM device and any memory location within the device may be addressed by an 11-bit address code from the microprocessor on the address bus (2^{11} = 2K). The RAM device can be both written into and read from so a read/write control signal (R/W) from the microprocessor instructs the memory to read the data from the data bus when low, or write data onto the data bus when high.

Supposing that this memory is too small, and that its size needs to be increased to double its present capacity. One way of doing this is shown in *figure 2*, where **chip select** ($\overline{CS}$) inputs to each memory device are used, with an inverter, to form a single controlling input to the two devices.

This input allows device 1 to be selected when low, but when high selects device 2. All memory devices are manufactured with some type of chip select input – one example that we have already met is the chip-enable/power down ($\overline{E}$) input of the 2167 static RAM seen in *Digital Electronics 17*.

To control memory device selection,

4



3. Memory map for the digital system in *figure 2*.

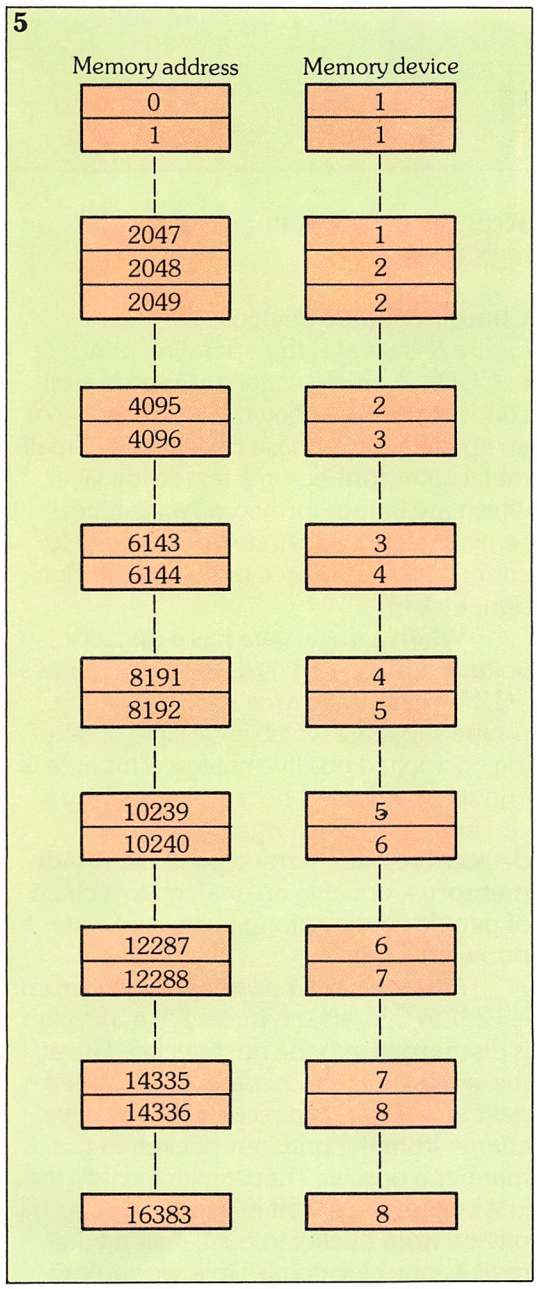
4. One method of joinig 8 memory devices to increase memory size.

an extra address bit is required, generated by the microprocessor, which allows selection of 4K of memory (2^{12} = 4K). The most significant bit of the 12-bit address code is used to select the memory device. We can see this in *figure 3* where a **memory map** of the digital system is illustrated: this shows that the lower half of the address space corresponds to the locations in memory device 1 and the upper half to memory device 2.

Where two memory devices still do not provide the required memory size, the chip select function allows for the combination of more than two memory devices – one possible method of joining eight memory devices is shown in *figure 4*.

In this example the three most significant bits of a 14-bit address code generated by the microprocessor are decoded, with an address decoder, to select one of the eight memory devices. The remaining 11 bits of the address code select the required location within the selected memory device. The address decoder could be a simple combinational logic device, decoding the 3-bit binary code to one-of-eight outputs; or it could be a ROM, with three inputs and eight outputs in which a decoding table is stored. *Figure 5* illustrates the memory map corresponding to this system.

5. Memory map for the digital system in *figure 4*.



This process of creating a memory with a larger address from a number of memory devices can be summarised in the following way: a number of memory devices may be joined to form a single memory device of address size LM (where L is the number of individual memory devices and M is the individual address size).

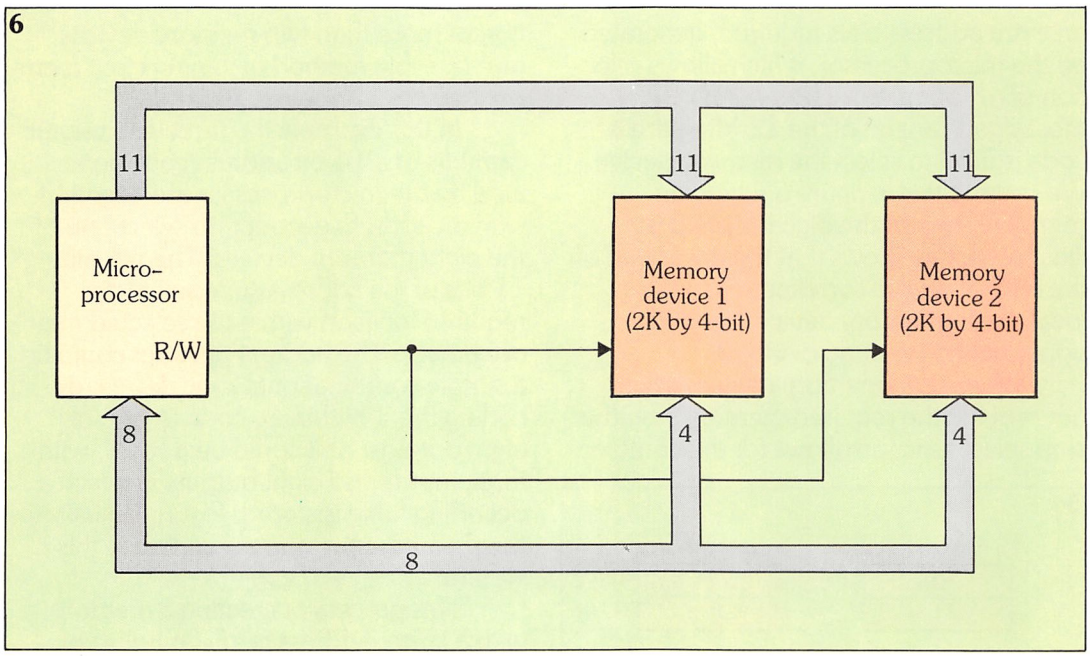
Larger word length

It may be that the memory devices used in a digital system are not individually capable of storing the required length of data word. *Figure 6* shows how two memory devices (each of which is a 2K by 4-bit device) may be joined to form a single memory of size 2K by 8-bits. To do this, the 11-bit address code generated by the microprocessor addresses both memory devices. Each 8-bit memory location used to store an 8-bit data word is made up from a 4-bit location of memory device 1 and a 4-bit location of memory device 2, both with the same 11-bit address.

We can summarise this process of creating a memory with a larger word length from a number of memory devices in a similar way to the previous summary: a number of memory devices may be joined to form a single memory of data word length LN-bits (where L is the number of individual memory devices and N is the individual device word length).

Larger address and larger word length

The third basic way of joining memory



6. Joining 2 2K by 4-bit memory devices to produce a digital system capable of storing an 8-bit word.

7. 32 16K by 1-bit memory devices joined to form a 64K by 8-bit memory.

devices to form larger memories in digital systems is simply a combination of the first two, and can be summarised in the same way: a number of memory devices may be joined to form a single memory of address size RM, and word length SN-bits (where $R \times S = L$, the number of individual memory devices; M is the address size of the individual devices; N is the individual device word length).

Figure 7 shows how 32 RAM devices, each of which has an address size of 16K and a word length of 1-bit, may be joined to form a single memory of 64K by 8-bits in a digital system.

Other memory devices

Although we have considered all of the important digital memory devices in depth, i.e. ROM, RAM, shift registers etc., there are a small number of others which deserve mention. Some of them, cores, disks, drums (all direct or random access memory types) and magnetic tape, paper tape, punched cards (all serial access memory types) have been discussed in relevant *Basic Computer Science* chapters and we needn't spend time on them here.

However, there are two other types of memory devices – the **charge-coupled device** (CCD) and the **bubble memory** – that have not yet been covered. Both of these use chips similar to those we have recently seen, and both provide serial access recirculating storage of the shift register type.

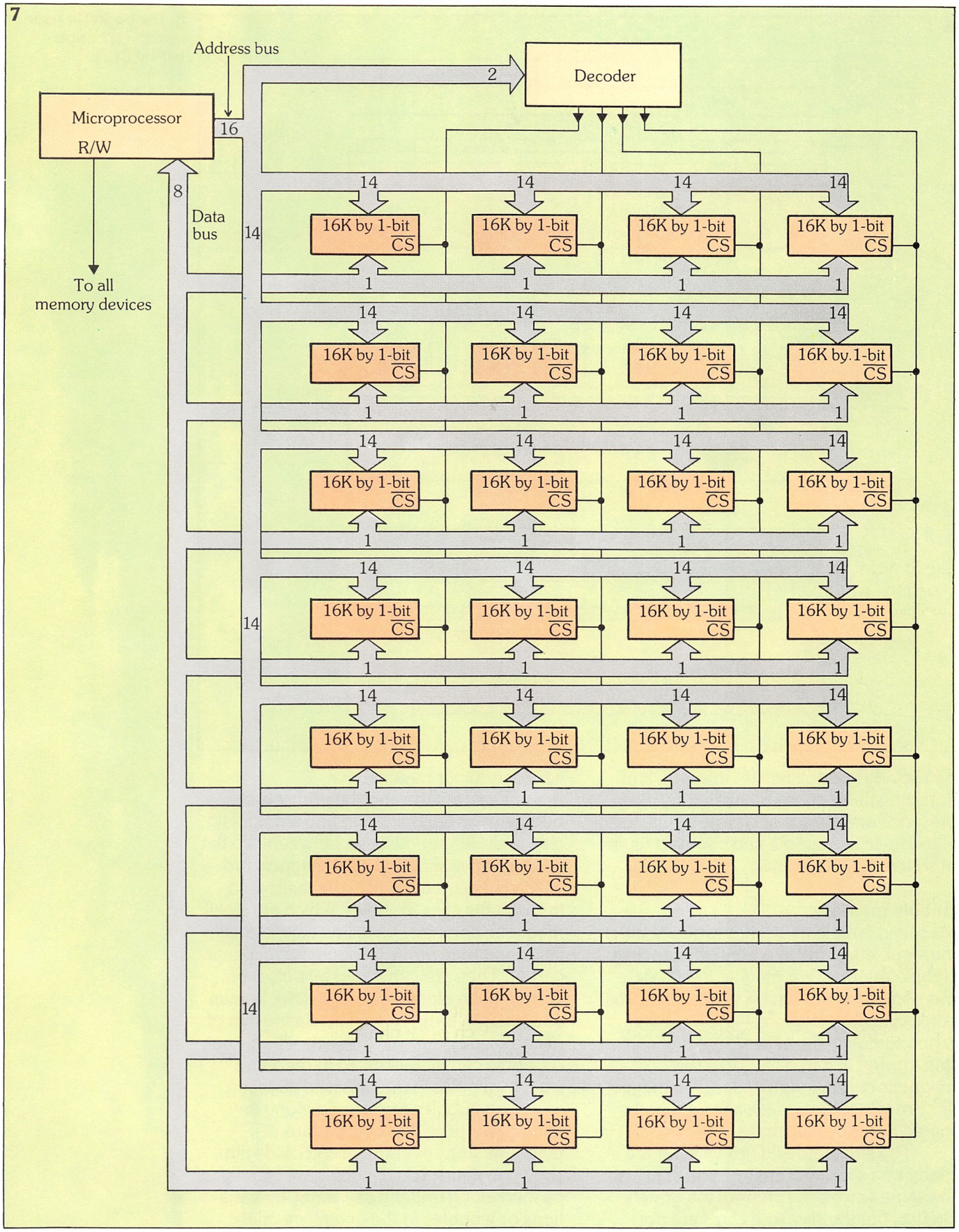
Charge-coupled devices

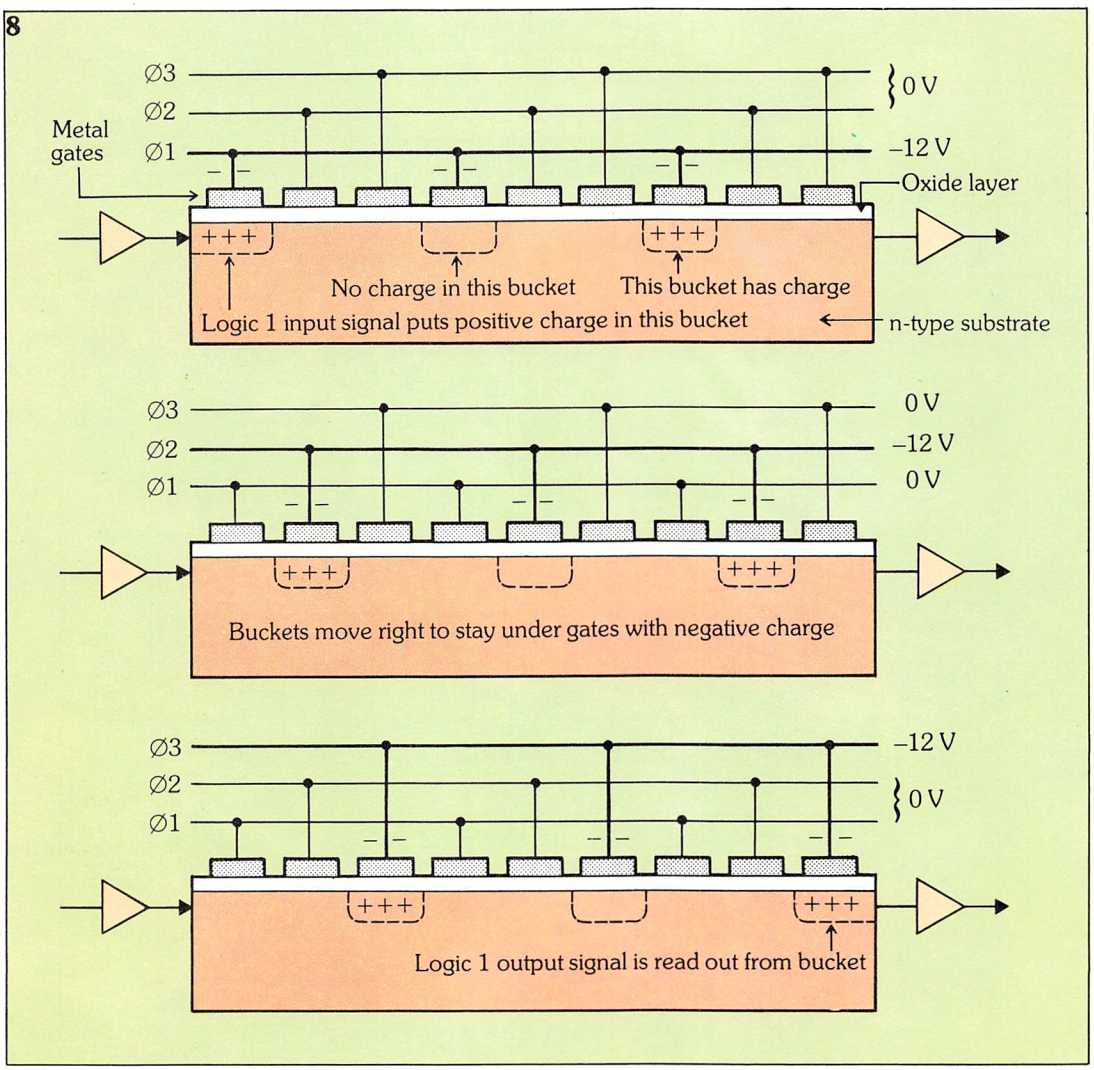
Figure 8 illustrates the operating principle of CCDs. A cross-section through a semiconductor chip is shown, at each phase of an applied three-phase clock signal. Small metal gates formed on a thin oxide layer, which are in turn formed on an n-type semiconductor substrate, are alternately connected to the three phases of the clock signal.

When a metal gate has a negative voltage applied to it (say, the clock signal's −12 V portion) an area is formed in the n-type substrate which is capable of holding an applied positive charge. This area is known as a **bucket** because of the way it can hold positive charge, hence CCD devices are often termed **bucket-brigade memory** – drawing an analogy to a chain of people passing along buckets of water to throw on a fire.

As each clock phase signal goes from −12 V to 0 V, any charge held in a bucket is discharged into the next bucket. But at the same time, another bucket is formed next to it, which can pick up any positive charge from the previous bucket. In this manner a positive charge injected into the first bucket on the left in *figure 8* will be passed from bucket to bucket along the row, as one clock phase follows another.

7


Address bus
Decoder
2
Microprocessor
R/W
16
8
Data bus
14
To all memory devices
16K by 1-bit
CS
1



8. The bucket brigade operating principle behind CCDs.

By providing amplifiers for injecting and detecting these charges, and letting the presence or absence of charge signify logic 1 and logic 0, a simple and compact type of shift register is formed.

Bubble memory

Magnetic bubble memories produce much the same effect but in a very different way. Bubble memory is a magnetic solid state memory, in which data is stored as microscopic cylindrical regions – the **bubbles** (approximately 5 μm across) – that can drift around in a thin film of garnet magnetic crystalline material. The magnetic garnet is grown epitaxially on a non-magnetic garnet substrate.

The substrate and grown layer are made as a slice, and cut into chips, in just the same way as the chips of integrated circuits, but the slice and chips are not semiconductor material.

Figure 9 illustrates the principle of bubble memory operation and a magnetic chip is shown sandwiched between two flat permanent magnets. These magnets provide a permanent vertical magnetic field through the chip. Bubbles, which are small magnetic domains, have inverse magnetic fields and may move through the magnetic garnet. They are generated by pulses of current in an aluminium conductor known as a **hairpin loop** over an insulating film of silicon oxide on top of the magnetic garnet. A bubble can be destroyed in the same way, by a pulse in the opposite direction.

The bubbles are steered along defined paths provided by a pattern of magnetic strips over the silicon oxide film, shown in *figure 10a*. Bubbles are literally pushed along under these strips, by variations or **wobbles** in the overall magnetic

field, generated by out-of-phase alternating currents through two sets of coils passing around the chip at right angles to each other (*figures 10b*).

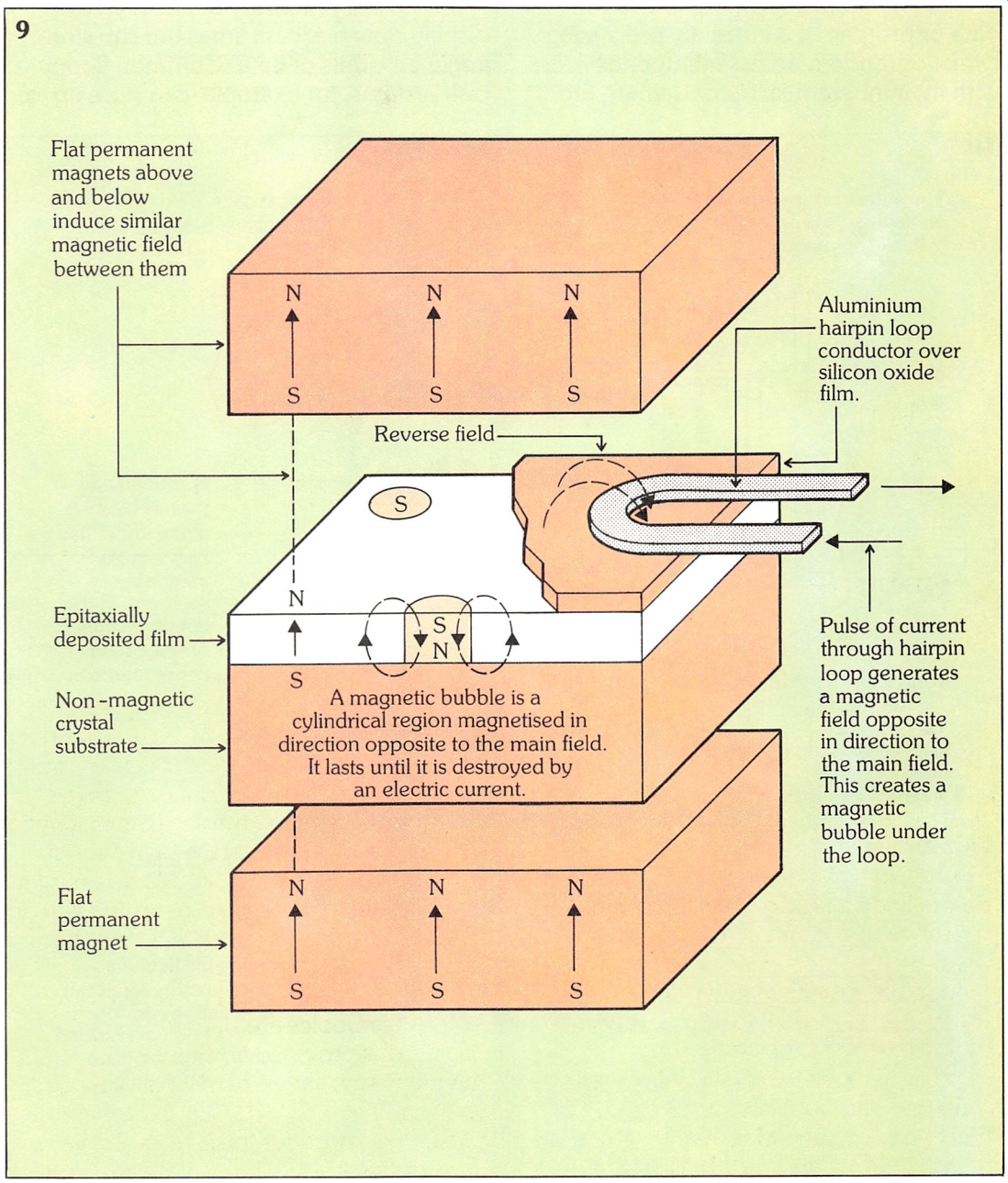
As the main field wobbles around a bubble, the segments of iron-nickel strips are turned into small bar magnets each with a N and a S pole. Each bubble moves to seek the nearest N pole of a segment, and so the paths formed by the segment keep the bubbles spaced out – one bubble per segment along the path.

The presence of a bubble indicates a logic 1 bit and absence means logic 0.

Each time the field wobbles, the bubbles all advance to the next segment, and bubbles on adjacent rows travel in opposite directions. The shifting frequency of this bubble memory shift register is therefore determined by the frequency at which the main field wobble occurs and hence the applied coil AC frequency.

Although bubble memory is similar in operation and use to CCD memory and other shift register type memories, it does have one important advantage – it is non-volatile and the bubbles of data are not destroyed when the power is switched off.

9. Bubble memory operation.



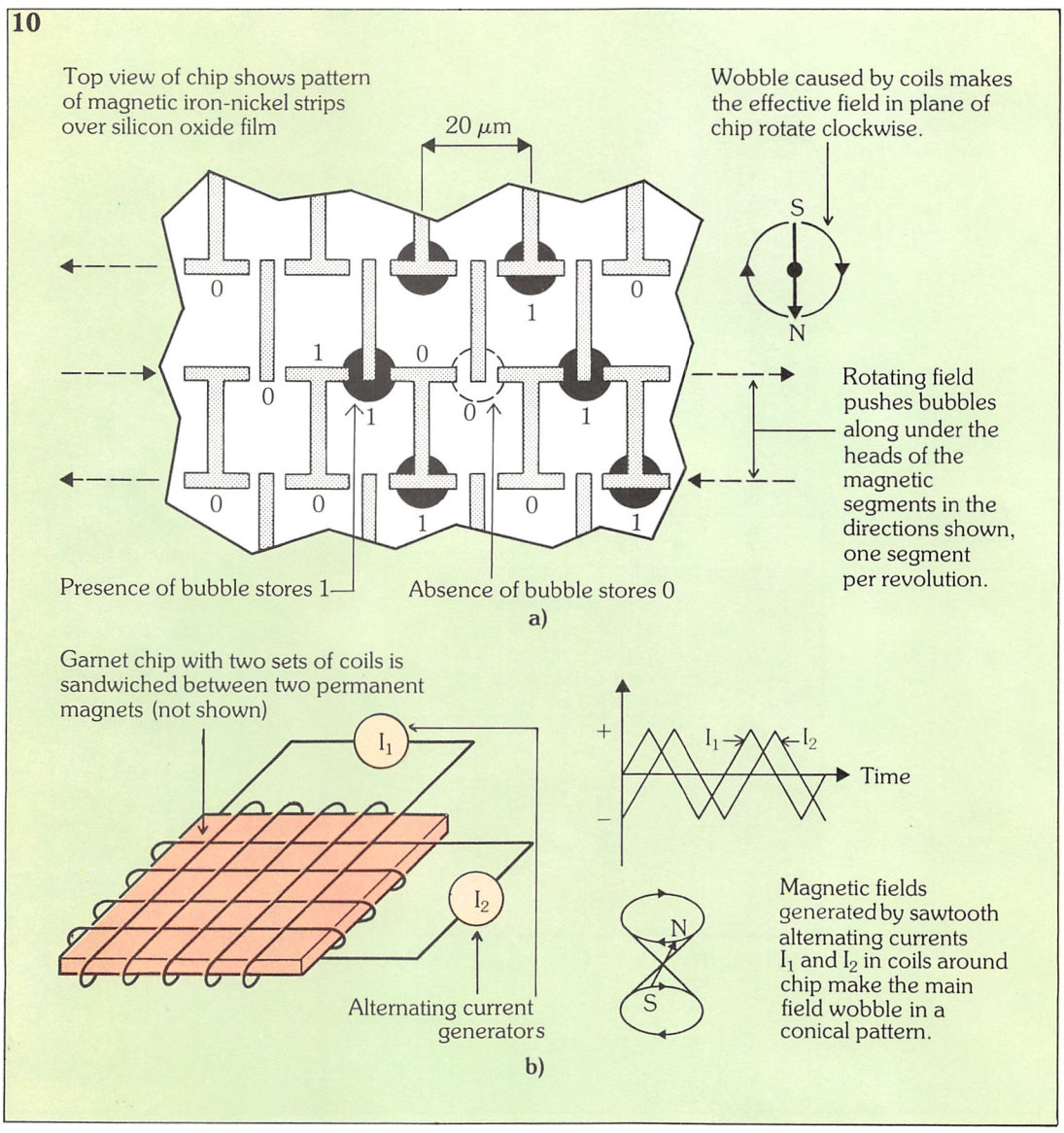
Summary of memory in digital systems

The memory used within digital systems depends largely on the cost, speed and availability of actual memory devices. Because of the wide differences in devices, digital memory – particularly in computer-based systems – tends to fall into one of two main categories: central or main memory, and auxiliary, mass or back-up memory.

Main memory is generally used by the CPU of a digital system as a fast access, small to medium sized store to hold the data or program it is currently processing. Direct or random access memory devices, with their inherent fast access times, are therefore ideal in such a memory. For example, modern microprocessor-based computers use semiconductor ROM and RAM in main memory – present ROM and RAM devices have access times between about 50 ns and 200 ns.

The size of the semiconductor main memory depends largely on the amount the user is prepared to pay, although memory devices are continually falling in price. Typical main memory sizes for personal computers are around 128K bytes to 256K bytes while home computers have memories around 16K bytes to 64K bytes.

Auxiliary memory devices typically display slower access times but can store larger amounts of data. Common floppy disk systems, for example, can store up to



10. (a) Bubbles are steered along defined paths by wobbles in the overall magnetic field; **(b)** this field is produced by out-of-phase alternating currents through two sets of coils at right angles to each other.

about 400K bytes on a single 3½" disk, with access times of about 3 or 4 ms.

Although we have seen that other memory devices exist, namely CCD and bubble memory, they each have their inherent disadvantages. The serial access storage methods of both types mean their access times are long compared with ROM and RAM semiconductor devices so they are not ideal as main memory devices.

The bubble memory does have a place, however, as auxiliary memory in portable digital systems. In such systems, disk memory with its high power consumption and magnetic tape memory with long access times are not suitable, so the non-volatility and reasonable access time of bubble memory is preferable.

In the future, we are likely to see more RAM and ROM devices with greater on-chip storage capacities and shorter access times. For example, 256K byte RAM are likely to be manufactured within the next year and 4 Mbyte devices are in an experimental stage. One of the most significant changes which may occur in the long term, is the improvement of E^2PROM and EAROM electrically alterable ROM devices. If they can be made easier and faster to write data into, large capacity non-volatile, cheap semiconductor memories will be made available.

Glossary

bubble	small domain of magnetised material in a thin film of magnetic garnet. The magnetic garnet is grown on a non-magnetic substrate
bubble memory	solid-state memory device operating on the principle that a bubble of magnetic material, polarised in the opposite direction to the overall magnetic field through the material, can move around in a controlled manner to form a shift register type memory
bucket	small volume of semiconductor material which is able to hold a quantity of electric charge
bucket-brigade	a number of semiconductor buckets in a line. Charge may be passed from bucket to bucket much like buckets of water are passed down a line of people to throw on a fire
charge-coupled device (CCD)	a semiconductor memory device operating by the bucket-brigade principle to form a shift register type memory
chip select ($\overline{CS}$)	terminal of a memory device which is used when more than one device is joined to form a single memory with larger address. The logic state at the terminal defines if the device is enabled or disabled
larger address	memory devices, say, RAMs may be joined together to create a memory with a larger address of size LM (where L is the number of individual memory devices and M is the individual address size)
larger word length	a memory with a word length of LN-bits can be made from L individual memory devices each with a word-length of N-bits
larger address and larger word length	a memory of address size RM, and word length SN-bits can be made from L individual memory devices of address size M and word length N-bits (where $R \times S = L$)
wobble	variation in magnetic field in a bubble memory, used to rotate the magnetic poles induced in the iron-nickel strips, thereby, moving the bubbles from strip to strip

Database systems

Introduction

A **database** is a collection of data files that is usually very large and complex. It is organised for a particular purpose, often in such a way that several users can access the data it contains.

With the widespread penetration of the small computer into everyday life, the term 'database' now includes address files controlled by home or personal computers. The smallest database can be configured in a system of one floppy disk drive, controlling at least two files connected together. Another example of a small database is a file with a primary and secondary index.

The use of databases on microcomputer-based systems is one of the expected growth application areas over the next ten years, but here we shall consider mainly minicomputers and mainframe type systems. These medium and large scale computers have good sized mass memories, which they can access directly.

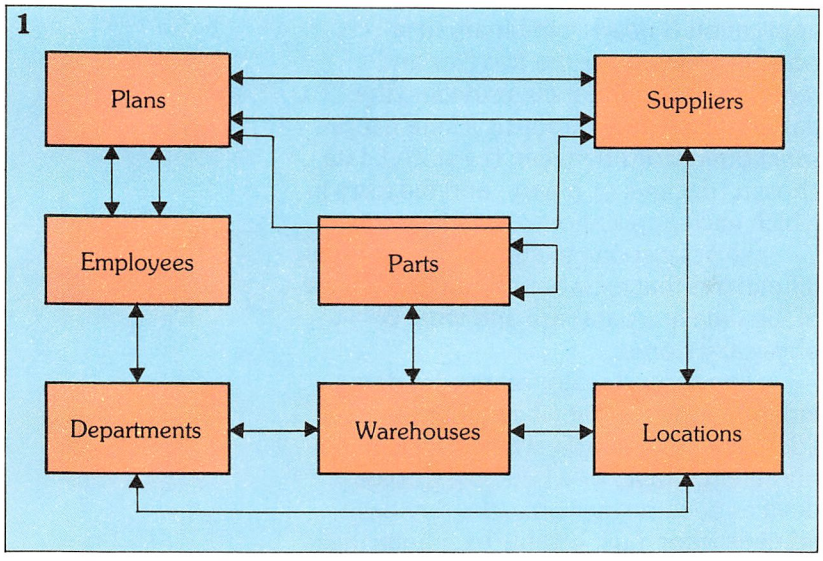
Another growth area is in knowledge-based or **expert systems**. These databases contain so much information that when you ask them questions their answers appear to be intelligent deductions. Expert systems can be used to great effect in medicine, for example, where a doctor may need to know many thousands of apparently insignificant diagnostic facts. The size of a computer's memory alone would make such systems valuable, but combined with the speed or processing power of today's machines, expert systems become invaluable tools for modern clinical analysis.

Below: computers at work in the office.

What is a database?

The database itself is a set of data that may be stored on disks, drums or on other secondary storage media, such as tapes. Supporting it are ordinary batch application programs which run the system by retrieving, updating, adding or deleting data from the base. Similarly, there may be on-line users who interact with the database, and they perform the operating functions. The commonest and most important function carried out by remote users is retrieval.

1. Operational data for a typical manufacturing company.



Where a database is required by a large organisation, a manufacturing company for example, an **integrated database** can be used which contains data for more than one user. The various portions of an integrated database overlap in several areas; the data in these shared areas can be accessed by many users.

Continuing the example of a manufacturing company, we can define its **operational data** as data which does not include input or output data, queues or any transient information. The operational data of this manufacturing company includes information on the plans it has in hand, data on the parts to be used in those plans, who supplies the parts, where the parts are to be kept, and so on. These **entities** form the basic database (*figure 1*).

Generally, there will be relationships between these entities which are shown as arrows in *figure 1*. The relationships are all bidirectional (parts are stored in warehouses and warehouses store parts, for example), and they form as much a part of the operational data as the information they relate. Somehow, therefore, these associations must be illustrated in the database: pointers may be used, but making entities adjacent to one another in the database suffices in some cases.

Figure 1 also illustrates some other points worth mentioning here. The arrow joining suppliers, parts and plans, for example, provides valuable information on who supplies which parts to which specific plans. That 'supplier A supplies part Z to plan 9', for example, tells us far more than 'supplier A supplies part Z', and 'part Z is used in plan 9'.

Each bidirectional arrow represents any number of transactions between entities; one relation might be 'employee P works on plan Q' and another could be 'employee M runs plan Q' – the two are subtly different.

Finally, there is the relation of an entity with itself. Parts, for example, may be built into standard subassemblies which will also be classified as parts.

Why use a database?

Databases provide centralised control for operational data, and as operational data is essential for the running of an organisation, so databases become vital assets. If a database that had not been either copied or backed up is destroyed (by a system crash or a fire, say), the company whose records it contained may well be out of business. The person or team responsible for centralised control is the **database administrator**.

It is the role of the database administrator, among many other functions, to reduce **redundancy** in stored data by identifying any applications that use essentially the same data and integrating the files.

It should be noted, however, that redundancy can speed up the access of information and that the reduction of a database to a non-redundant entity will render the system unusable.

The possibility of **inconsistency** can be almost eliminated from a database in the same way. Consider two identical files:

if one file is updated then the two files will not agree. The use of one entry – by removing the redundant file – thereby removes the inconsistency. As in all large systems a balance must be sought between the maintenance of consistent relations and sufficient redundancy to make access times acceptable.

Other tasks allotted to the database administrator includes **security** (who has access to what), and **integrity** of the database (ensuring that it contains only accurate data). These aspects of database management will be discussed later.

What goes into a database?

The **data administrator** is a person or team concerned with the data in the database, rather than the details of its implementation. The data administrator must establish and maintain precise definitions of each entity, attribute and relationship within the **data model**. The data model can be thought of as being one single view of the database contents appropriate to one application – the database may comprise several integrated data models.

Synonyms and homonyms that have become established by use must be recorded. For example, an 'account' to a marketing department often means 'customer' to a finance officer. This process of data analysis generates more data – data about the data. This **meta-data** must itself be organised and updated when necessary: it forms a data dictionary system.

The dictionary should include the minimum number of words necessary to define each document uniquely, and these are known as **key words**. Using these key words, the dictionary acts like a thesaurus: when users request a file, they cite a number of words which they think sufficient to identify the subject sought; the system replies with the document in which all those key words appear.

Data independence

Possibly the most important objective of database systems is to provide **data independence**. Most applications today depend on the way that data is organised on secondary storage, and on the way that the data is accessed. It is impossible to change the storage structure of a data dependent file, or the method of access without making major modifications to the application.

This is undesirable in database systems for two major reasons. First, different applications need different views of the same data: for example, where application A sees customer balance in decimal, but application B needs it in binary. It is possible to integrate the two files and remove the redundancy, providing the database software performs all the necessary conversions between the stored data chosen (decimal or binary) and the form in which each application wishes to see it.

This is just one example of the kind of differences that exist between a user's view of the data in a database and what is physically stored.

The second major reason for data independence is that when an enterprise adopts new standards, when application priorities change, or when new storage devices become available, the database administrator must be able to change the storage structure or the access strategy in response.

Data alterations

Later we will consider the **architecture** for database systems that provides data independence, but first let us look at the types of change that a database administrator might want to make. These are changes to which the applications we perform should be immune.

In most modern systems the logical record of an application is identical to its **stored record**. (A stored record is an identifiable collection of associated stored fields (the smallest stored unit of data in the database)). These can be distinguished by type and occurrence. A stored record occurrence would contain an occurrence of each of the following stored fields: part number, part name, part weight and part colour. The association between them is that they all contain information about one particular part. A stored record type is a set of occurrences for each distinct part.) In a database system, however, the database administrator may make changes to the structure, i.e. to the stored fields and records, while the logical fields and records

do not change. This is data independence. For example, a part's weight could be stored in binary to save space, but a COBOL type application might see it as a character string (for example, EBCDIC or ASCII).

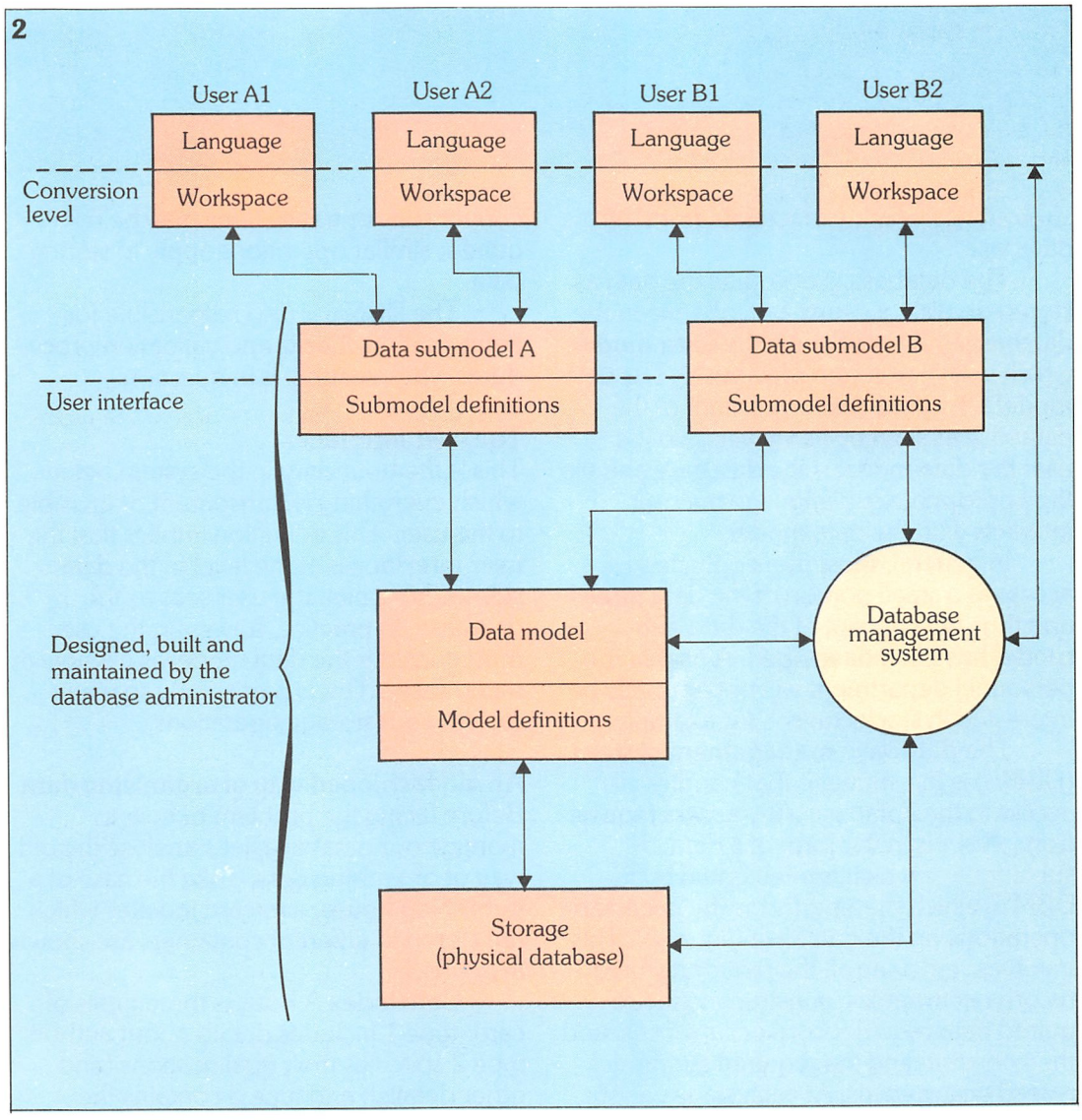
Using this series of stored fields and records, a database should be able to grow without applications being outwardly affected. This is the result of data independence. New types of field can be added to the database and these will be invisible or transparent to all previous applications.

In addition, it should be possible to add new types of relationship to the database. How this is done is dependent on how these relationships are presented and this is a property of the type of architecture used.

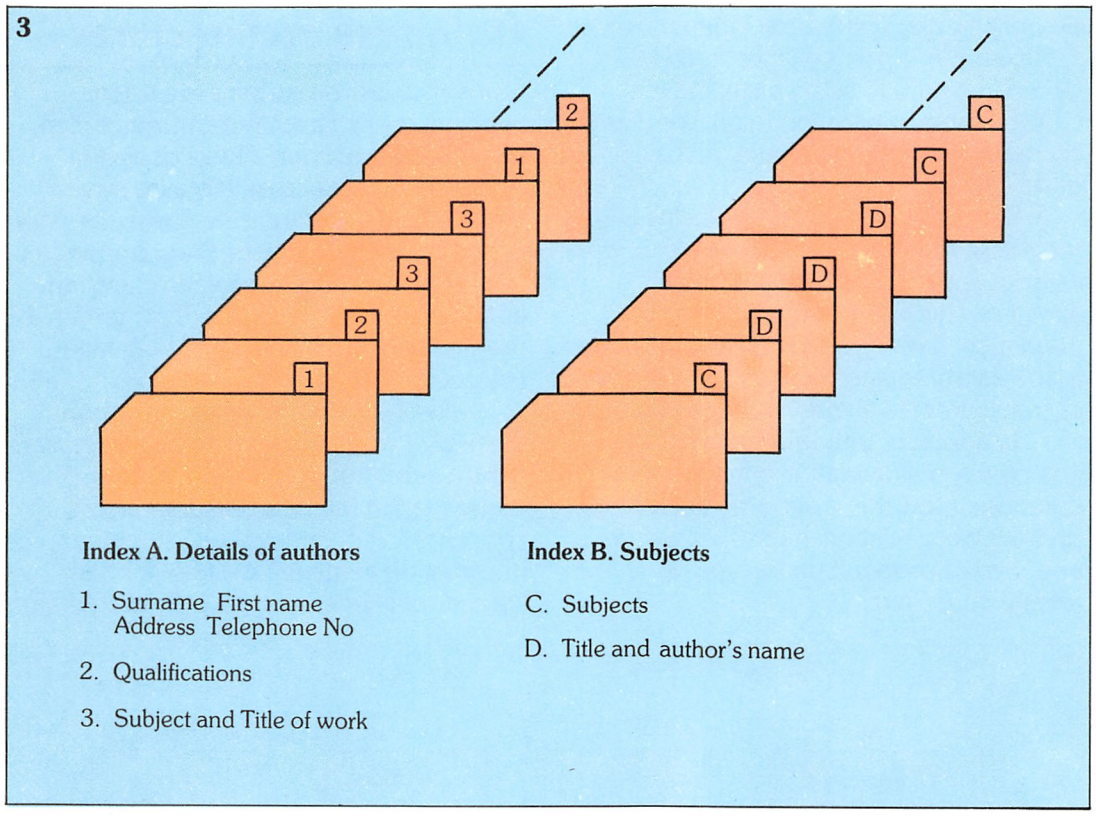
Database architectures

Users of database systems are usually applications programmers or remote terminal users – account clerks or design engineers, for example. All users have a language for communicating with the database: for the applications programmer it will be a conventional programming language, such as COBOL or PL/1; terminal users may have a specialised language, tailored to their individual needs.

The important thing to note about these user languages is that they all contain a **data sublanguage** (DSL). This is embedded in the host language and is concerned with retrieval and storage of information in the database — it often consists of little more than calls to standard



2. An architecture for a simple database.



3. Organising a card index.

subroutines which extract data from the database.

The database is of course the data as stored on disk or drum, but inbetween the database and the user sits the **data model** which was mentioned previously. The data model is the information content of the database as seen by its viewers. To the user the data model *is* the database and via the appropriate sublanguage the user interacts with the data model.

In general, most users only need access to a small portion of the data model and thus the concept of the **data submodel** has been developed. A user in the personnel department will not normally be interested in stock control, for example.

The **database management system** (DBMS) is the software that handles all access to the database. Any access request from a user is in the form of a data sublanguage which is intercepted by the DBMS, which then performs the necessary operations on the database (*figure 2*). This involves retrieving all the required stored record occurrences, constructing the required data model record occurrences, and then constructing the required submodel record occurrences. At each stage various conversions or translations may be required; similar operations apply to storing data.

The DBMS is also responsible for authorisation checks and validation procedures when security is in question.

The user interface

This is the boundary in the system below which everything is transparent or invisible to the user. This definition implies that the **user interface** is at the level of the data submodel, which the user sees as the database, in practice, however, the user must consider the data model as a whole in many cases. This is particularly so when carrying out storage operations.

An old fashioned way of organising data

Before facing the problem of how to manage our database, let's analyse the old way of organising data. Take the case of a publishing house: two card indexes which are stored in different containers are shown in *figure 3*.

Card index A houses three types of card: type 1 includes details about authors; type 2 specifies their qualifications (and other details); and type 3 contains the

4. Sample data.

4

I_x	NAME	QUAL	SUBJECT
I_1	Shore	30	America
I_2	Worth	10	Poetry
I_3	Blake	20	Poetry
I_4	Cooke	20	America
I_5	Adams	30	Zoology

By comparison, card index B contains a type C card for every subject, followed by as many type D cards as there are works published on that subject. On the D cards are the authors' names, together with the titles of their relevant works. This is redundant information, however, as it already appears on the type 3 cards of index A. In order to eliminate this redundancy a GO TO FILE A, AUTHOR X, TYPE 3 marker should be placed on card D, enabling the index user to find the required information, without having it needlessly repeated in index B.

The drawback to this method, however, is that the physical layout of card index B does not now correspond to its logical organisation: the D cards no longer follow the C cards and are now somewhere else.

5. Factoring out the subjects saves space.

5

AUTHOR FILE

I_x	NAME	QUAL	SUBJECT POINTER
I_1	Shore	30	
I_2	Worth	10	
I_3	Blake	20	
I_4	Cooke	20	
I_5	Adams	30	

SUBJECT FILE

SUBJECT
America
Poetry
Zoology

subjects and titles of their works. Obviously, type A cards must remain in alphabetical order according to the authors' surnames. Each author will have a type 1 and type 2 card, but there may be many type 3 cards, depending on how prolific the writer is.

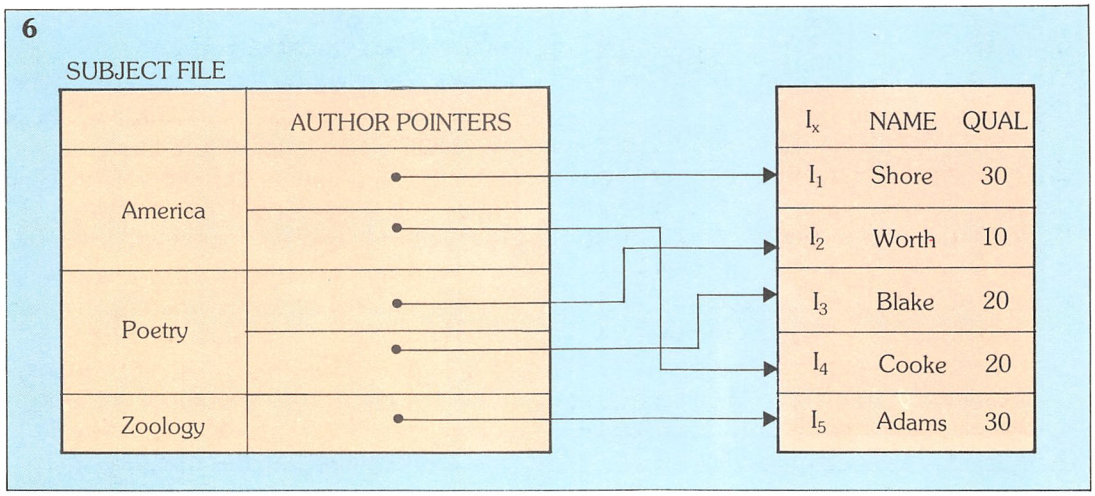
The type 2 and 3 A cards should also contain an indication of their connection with the author, or first card. This is to reposition them should they become misplaced. This identification could be the surname of the author, or the author's unique identification number, clearly marked on type 1 cards.

The physical organisation of the card index is the same as its logical organisation: First is a type 1 card for author Able, followed by type 2 and then type 3 cards which are also ordered, perhaps according to publication date. Using small markers or tabs for each type of card, this index can easily be handled.

Database representation of sample data

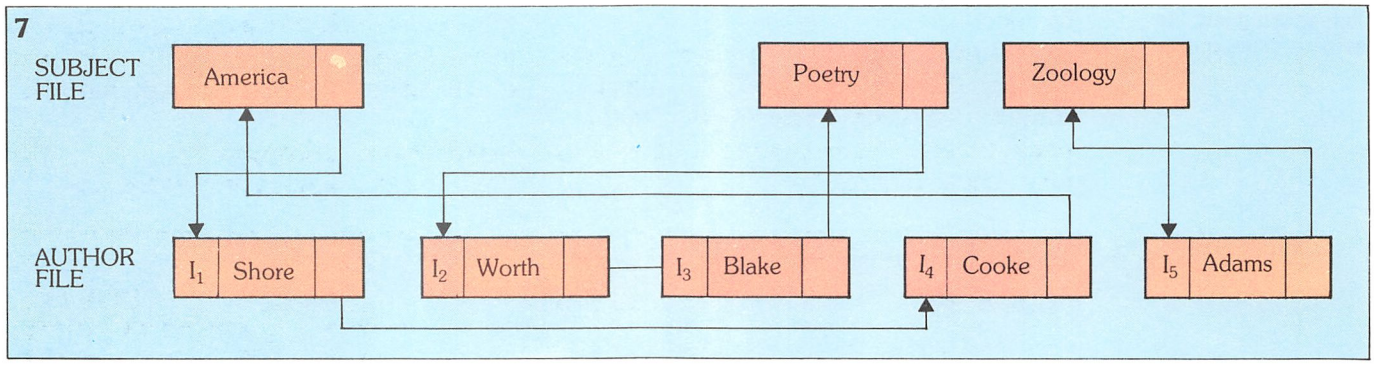
We can see that the card index is an unsatisfactory way of storing a large amount of data: its redundant and illogical nature would soon consume its allotted storage space. Let's consider instead some of the ways in which this data could be represented in a computerised database, at the level of the stored record interface.

Sample data consisting of information about five authors is listed in *figure 4*. For each author we need to record an identification number (I_x), a surname, qualifications (some arbitrary number), and a subject. (For simplicity we shall ignore at present individual works.) Each file is sequenced by the access method on its



6. Indexing on subjects.

7. Pointer chains in a database.



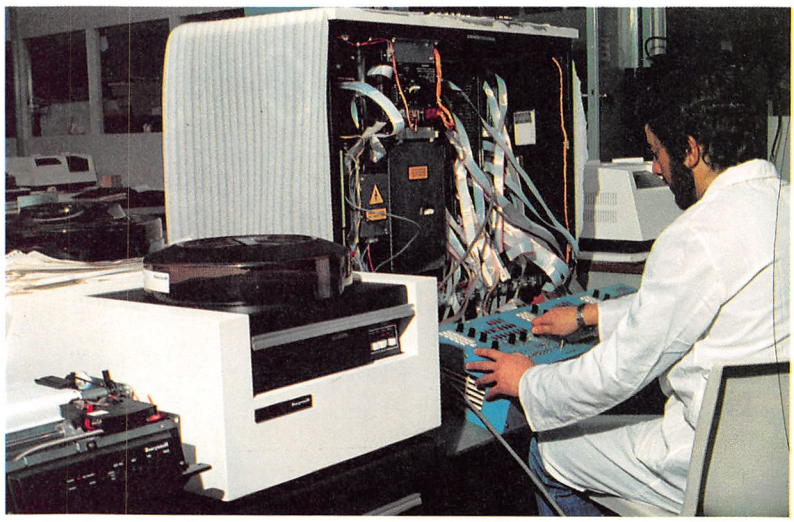
identification number or **primary key**.

The simplest method of presenting this data is in a stored file containing five stored record occurrences, one for each author. Suppose, however, that we have 10,000 authors writing on ten different subjects. Assuming that the space taken up by a pointer is less than that needed for a subject name, we can save a large amount of storage if we factor out the subject titles and display the database as two stored files. The only advantage of this presentation (*figure 5*) is the saving in space.

If the query 'find all the authors on a particular subject' is important, the database administrator may wish to present the database as in *figure 6*. Here, there are also two stored files, but this time the pointers are from the subject file into the author file. This presentation is worse, though, for queries regarding all the attributes of a given author.

An interesting point to note about *figure 6* is that the subject file acts as an **index** to the author file. It is controlled by the DBMS and it is known as a **dense secondary index**: dense means that it contains one entry for every stored record occurrence; and secondary means that it is an index on a field other than the primary key.

If we combine the two presentations so that each stored record occurrence (author or subject) contains only one pointer, we can design a chain of pointers. For example, each subject points to the



Above: testing the Honeywell DPS4. (Photo: Honeywell).

first writer on that subject, who then points to the second writer, who points back to the subject (if there are only two authors) (*figure 7*).

The advantages of this model are clarity and ease of making changes to the indexing. Its disadvantage is that to reach the last author on each subject you must follow the chain and access all the other authors, too. If each access involves a disk seek operation, the time to reach the *n*th author may be quite high.

Using a secondary index it is possible to design an **inverted organisation** where every field type is indexed and has pointers (*figure 8*), but this is very costly in terms of storage.

A simpler design is illustrated in *figure 9*. Here, there is one stored file containing three stored record occurrences, one for each subject. This arrangement is termed a hierarchical organisation.

Characteristics of data models

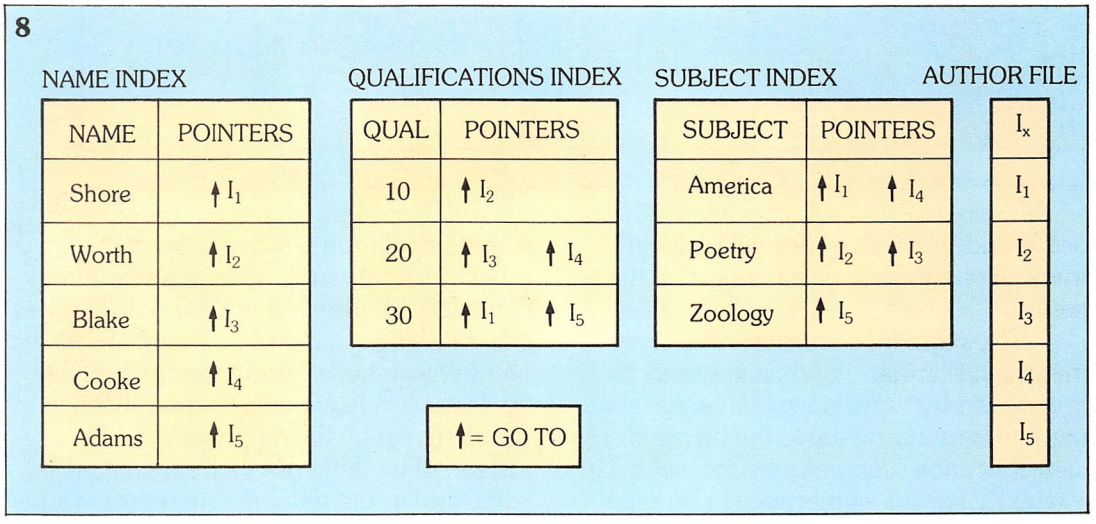
There are three general approaches to designing the data model: the **hierarchical approach**; the **network approach**; and the **relational approach**.

At present, no large scale implementations of relational databases exist, so there is no practical experience of the effectiveness of this approach. It is, however, the most promising of the three approaches and much effort is being directed into this area. There are many examples of the other two approaches, however, which exist as compromises between what is desirable at the user interface and what can be efficiently implemented.

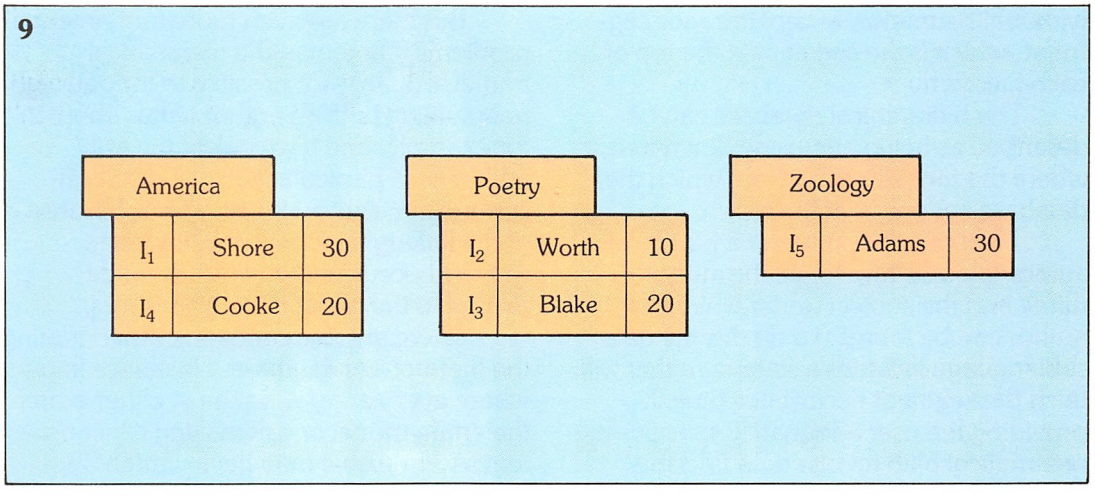
The hierarchical approach

This approach is used in many existing

8. Inverted organisation.



9. Hierarchical organisation.



database systems, including IBM's Information Management System (IMS). It was developed with the storage structures that existed when data processing was carried out using sequential media. Because of this, there is only a small distinction to be made between the data model and the storage structures themselves.

Returning to our authors' file, it is possible to present this data in a hierarchical model where subjects are superior to authors: this was shown in *figure 9*. This model shows the user three hierarchical occurrences, one for each subject; each occurrence consists of one subject **segment occurrence** for each author in the record.

The segment occurrence is usually the smallest amount of data that may be transferred by one data sublanguage statement. It is fundamental to the hierarchical view that each segment must be viewed in context for its full significance to be seen. No segment occurrence may be seen without its superior, except the **root segment**, which is the segment at the top of each hierarchy.

The hierarchical database can be described as being an upside down tree, where the root segment, from which the database springs, is at the top.

To retrieve an entry for a particular author the user must state the author's name and the subject under which that author can be found. To do this the data sublanguage includes a statement that will fetch the segment occurrence directly, providing the user adequately specifies the hierarchical path involved. In IMS this statement is 'GET UNIQUE'.

Another IMS sublanguage statement, 'GET NEXT', allows the user to jump forward to the next segment occurrence in sequence. However, this may not be the segment that the user wants.

A major advantage of the hierarchical system is that it is a very natural way of modelling a structure from the real world. Difficulties arise, however, when an attempt is made to use data sublanguages. Consider these sample queries based on the files we have already built (*figure 10*).

Although the original queries are symmetrical (one is the reverse of the other), the data sublanguage procedures necessary to answer them are not. This illustrates the main disadvantage of the hierarchical model: unnecessary complexity. Database users spend time solving problems set by the database model, instead of working out their original problems, and as the database increases in size, the hierarchy becomes more complex.

But there are even more fundamental problems inherent in the hierarchical framework. It is not possible to introduce a new subject until an author writes about it, for example; and if we delete the only author on a particular subject, data concerning that subject is lost. This is because of the linking together of subordinate segment occurrences, which is fundamental to the hierarchical philosophy.

There are also problems with updating the hierarchical database. Unless we know where every entry is, we must either search the entire model or risk making it inconsistent. Author Able may have written 39

10

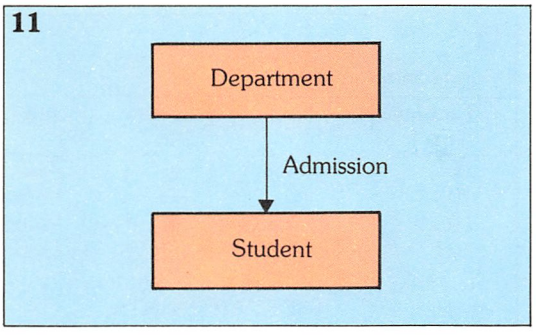
Question 1: Find subjects that author I_2 writes about.

Get unique author with $I_x = I_2$
Next: Get next subject for this author.
Subject found? If not, exit.
Print subject.
Go to Next.

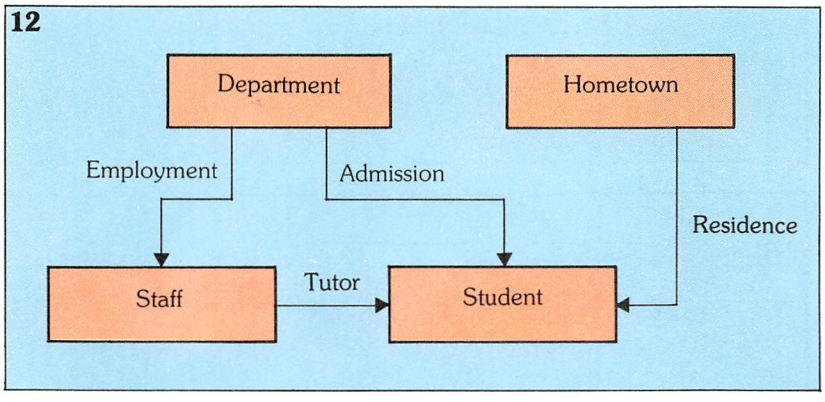
Question 2: Find author numbers for authors who write about America.

Get to start of data.
Next: Get next author.
Author found? If not, exit.
Get next subject for this author, if subject 'America'.
Subject found? If not, go to Next.
Print I_x
Go to Next.

10. Sample queries on the hierarchical model.



11. A CODASYL set structure diagram.



12. Structure of sets.

books in one place, but 40 in another. This is again a fault of the disparity between the logical and physical designs of hierarchical databases.

The network approach

A typical example of the network approach is the system proposed by the Database Task Group of CODASYL (**co**nference on **da**ta **sy**stems **l**anguage). The meaning of the word 'network' in this context is quite different from that of the communications network.

The fundamental concept in the CODASYL system is that of a set. A **set type** is the association between two separate record types: one of the record types is the owner of the set and the other is the member.

An occurrence of a record type is a specific set of values that satisfy the definition of that record type. For example, if the record type student (*figure 11*) is defined to include name and length of study then Jones 2 and Smith 3 are two occurrences of student. An occurrence of a set type consists of one occurrence of its owner with or without any members.

The rules defined by CODASYL make building sets fairly easy. For example, a record type that is a member of one set can be an owner of another (which allows hierarchies to develop); and a record type may be either an owner or a member of more than one set (which introduces non-hierarchical networks).

These ideas can be seen in operation in *figure 12*, where 'student' is a member of three record sets, 'department' is an owner of two sets and 'staff' is a member of the set 'employment', but an owner of the set 'tutor'.

The major disadvantage of the network model is that it too closely mirrors the structure of the storage. Consequently, the user must be aware of the chains that do and do not exist. There is a risk here that the user will become locked into a particular storage structure, contrary to the aims of data independence.

The relational approach

In comparison with hierarchical and network based systems, relational databases are fairly easy to use. A simple analogy is to compare the difference between COBOL and the assembler language to write programs. In assembler, you need to know the bits and bytes that make up registers and data addresses, but with COBOL a programmer uses very high level language to make English-like statements.

When using other database techniques, you need to know – as we have discovered – *how* the data is structured and *where* to find it. In relational databases all you need to know is *what* you are looking for.

The relational approach treats all data as two dimensional tables, with rows and columns. To manipulate or access data there are three basic operations: JOIN, PROJECT and SELECT.

To obtain information from these two files (*figure 13*) the user orders a JOIN, which collates the data in the tables through a common field, department number (*figure 14*). Then, using PROJECTION, the user can decide which columns need to be displayed (*figure 15*), and finally in a SELECTION the user chooses which rows to see (*figure 16*). In addition, retrieval conditions can be set, for example, 'salary over £5000' or 'name starts WI'.

The relational approach is based on the mathematical theory of relations (*figure*

17); the results of this theory can be applied directly to designing the data sublanguage. Very simply, the relation is the whole file, the records are known as **tuples** and the fields are called **domains**.

Figure 18 shows a relation called PART, which is defined on domains Px (part number), NAME, COLOUR and WEIGHT. The domain colour, for example, is the set of all valid part colours. The relation consists of six tuples, or rows, and four domains, or columns. From this relation we can deduce some fundamental points and assumptions about relational databases in general.

1) No two rows (tuples) are identical.
2) The ordering of rows (tuples) is insignificant.
3) The ordering of columns (domains) is insignificant (providing we always refer to domains by name and not by position).
4) Every value in a relation (each domain value in each tuple) is an atomic, or non-decomposable, data item (e.g. a number or a character string).

To expand on the last point, at every position in a relational table there should be one and only one value – never a set of values. A relation satisfying this property is said to be **normalised**. Consider *figure 19*: BEFORE is unnormalised, but AFTER is normalised. All that it means is that in every tuple a supplier supplies a certain part in the indicated quantity.

The relational model of a database,

13

(a) EMPLOYEES

Employee No.	Name	Dept. No.	Job	Salary
101	Smith	50	Assembler	6000
103	Jones	50	Packer	5000
105	Adams	55	Programmer	8000
107	Wilson	62	Clerk	5000

(b) DEPARTMENTS

Dept. No.	Dept. Name	Location
50	MFG	Slough
55	ACCTG	London
62	LEGAL	London

14

Dept. No.	Employee No.	Name	Job	Salary	Dept. Name	Location
50	101	Smith	Assembler	6000	MFG	Slough
50	103	Jones	Packer	5000	MFG	Slough
55	105	Adams	Programmer	8000	ACCTG	London
62	107	Wilson	Clerk	5000	LEGAL	London

15

PROJECT NAME DEPT NAME & SALARY

NAME	DEPT NAME	SALARY
Smith	MFG	6000
Jones	MFG	5000
Adams	ACCTG	8000
Wilson	LEGAL	5000

13. Two files in a relational database.

14. Files joined on department number.

15. Using a projection the user decides which columns are to be displayed.

Left: EPSON HX-20 portable computer. (Photo: EPSON).

then, is a collection of time-varying normalised relations of assorted sizes or degrees. We specify 'time-varying' to allow for insertion and modification of tuples and domains.

16. During a selection the user decides which rows to see.

16

SELECT SALARY OVER £5,000

NAME	DEPT NAME	SALARY
Smith	MFG	6000
Adams	ACCTG	8000

17. Mathematics of the relational theory

Given sets D_1, D_2, . . . D_n (not necessarily distinct), R is a relation on these n sets if it is a set of ordered n-tuples d_1, d_2, . . . d_n such that d_1 belongs to D_1, d_2 belongs to D_2 and so on.

Sets D_1, D_2 . . . D_n are called the domains of R, and the value of n is called the degree of R.

17. The relational theory.

18. The relation PART.

19. Normalisation.

18

PART

Px	NAME	COLOUR	WEIGHT
P_1	Nut	Red	10
P_2	Bolt	Green	16
P_3	Screw	Blue	16
P_4	Screw	Red	14
P_5	Cam	Orange	09

19

BEFORE

I_x	PQ	
	P_x	QTY
I_1	P_1	3
	P_2	2
	P_3	4
I_2	P_1	1
	P_2	1

AFTER

I_x	P_x	QTY
I_1	P_1	3
I_1	P_2	2
I_1	P_3	4
I_2	P_1	1
I_2	P_2	1

Security in databases

The protection of the data in a database against unauthorised disclosure, alteration or destruction is becoming increasingly important. It is not within the scope of this series of articles to discuss the ethical and social aspects of this problem, but we will examine measures and safeguards that can be taken, particularly regarding the hierarchical and network databases.

Suppose that a database includes a relation EMPLOYEE, defined on domains EMPx (employee number), NAME, ADDRESS, DEPTx (department number), SALARY and ASSESSMENT (manager's evaluation of performance). We can imagine many situations where database administrators would want to define reasonable levels of access to this relation for various categories of user.

An accounts clerk, for example, may need to see the employee number and salary domains, and within any one tuple the clerk may be allowed to alter the salary value. However, we decide that this should only take place between the hours of 9.00 a.m. and 5.00 p.m. and then only via a terminal in the payroll office.

This one example provides some idea of the range and flexibility that a general purpose security scheme must have. The cost of implementing all of these security features would probably be extremely high, both financially and in terms of system performance degradation. A database administrator must choose measures which are particularly effective for his or her particular database.

Identification and authentication

It is the function of the database management system to identify and authenticate users who need access to the database, or to areas of the database. Authorisation is usually the responsibility of the database administrator, who defines to the system the operations that each user is allowed to perform. Users, in turn, then find out from the administrator the procedure for identifying themselves to the system.

One method of user identification might include the entering of an operator number followed by some password, supposedly known only by legitimate users of

that operator number. Other methods of user identification to a database include the use of bar-coded and laser-etched or holographic cards. Remote terminals of the future may also identify users by finger or voice print.

One procedure for authenticating users is as follows. After the user has been identified, the system supplies a pseudo-random number, x. The user then performs some simple mental transformation T on x and sends the result y back to the system. The system knows the answer and can clear the database user.

Any would-be infiltrator will see at most x and y, from which it is fairly impossible to work out exactly the nature of the transformation, even for a simple calculation such as the following:

$$T(x) = (\text{sum of 1st, 2nd, 3rd digits of } x)^2 + \text{hour of day}$$

In theory, the DBMS should check each data sublanguage operation as it is issued, to see whether it violates security restrictions. This is not really practical, however, and access constraints are more logically associated with data than with users. These constraints have to be specified in the data definitions, rather than as part of the user definition.

Security in hierarchical networks

All access from programs to hierarchical databases is through an appropriate **program communication block** or PCB. This confers two levels of protection: first, the program cannot access 'non-sensitive' segments at all (i.e. those segments it can have no action upon); and secondly, the program is constrained to those operations predefined in the PCB.

Further control over access is conferred by specifying the terminals that can use restricted applications programs, and also by specifying which terminals can give certain system commands.

Security in network databases

In a network database, all access from a program is via a **sub-schema** (the equivalent of the data submodel in hierarchical databases). Like the PCB in hierarchical databases, this immediately confers automatic security on hidden data. Unlike the hierarchical security system, however, the sub-schema cannot detect violations before the execution order has been given. Hierarchical databases can, in theory, detect some violations at translation time.

Network databases can have locks and keys: a random character string may be chosen as the key to a certain procedure or file. A program that wishes to access that file or carry out that action must supply a matching key. If the keys are equal, as compared by the DBMS, the operation will proceed; if there is a mismatch, the operation is suppressed and a privacy breach or error status message recorded.

Bypassing the system

These techniques are not sufficient to protect the database against an infiltrator who attempts to bypass the system. The most obvious example is when an infiltrator removes part of the database by stealing a disk pack for example.

Probably the most effective way of safeguarding against theft is to use scrambling techniques, or **encryption** and **privacy transformations**. The idea behind scrambling is that the work involved in unscrambling stolen data should far outweigh the potential advantage to be gained by doing so.

Several scrambling techniques exist, but the objectives are all the same. The inventor of the method, holding matching scrambled and unscrambled texts, should be able to determine a **privacy key**. This is a special group of characters algebraically entwined with groups of characters throughout the database. It is then a fairly straightforward task to translate the scrambled garbage into meaningful data.

Integrity

Maintaining the intergrity of a database means protecting it against invalid (as opposed to illegal) alteration or destruction. Integrity is thus distinct from security. There is a limit on how far this can be carried, but the data in a database should be as accurate as possible for as long as possible.

It is difficult for the system to check the accuracy of every single item entered into the database, but it can certainly check

for plausibility. For example, there may be no way of checking whether an employee worked 33 or 35 hours in one week, but an input value of 350 would obviously be wrong and the system should reject it.

Like security constraints, integrity restrictions are held in the system directory of relations and may be thought of as an extension of the data model definitions.

There are two basic problems in maintaining integrity: first, data validation; and secondly, the consistency problem.

Data validation

By definition, the primary key of any relation is unique. The DBMS must therefore reject any attempt to introduce a tuple whose key value is a duplicate of one that already exists.

Values in a particular domain, or domain combination, may be required to lie within certain bounds. For example, values of EMPLOYEE AGE must be greater than 15 and less than 66. In general, then, to enforce uniqueness and validity constraints, the DBMS must monitor PUT and UPDATE operations.

As regards consistency, the major culprit is the DELETE operation, so this also should be monitored.

Database utilities

Certain other utilities are necessary to help maintain the integrity of the database. Every transaction on the database should be recorded on a system journal, usually a tape. The journal will record a transaction number, a time stamp, the identification of the user who carried out the transaction, the address of the changed data and its before and after values. This should not really be classed as an extra utility, as it is an essential part of the system at all times. It is a central component of the DBMS.

Dump routines take back-up copies of the database and while they are doing so they usually need exclusive control of the data they are dumping. The importance of back-up cannot be over stressed; the database is a company's most vital asset.

Recovery routines are used to restore the database to an earlier state after a loss of integrity has been detected. Using a back-up copy of the database, together with the system journal, these routines reconstruct a new copy of the data as it was just before the error occurred.

Expert systems

Knowledge based or expert systems are essentially programs that manage highly specialised databases. Conceptually, they are a collection of assertions gathered in a base, rather than just 'data' bases.

The expert database contains facts, assumptions and heuristics, or guidelines – rules of thumb. Using this database is another part of the expert system, the inference engine, or processor, where all the reasoning takes place.

PROLOG (for **pro**gramming **log**ic) is a compiler put together by the University of Edinburgh. The programmer users of PROLOG think that they are programming a machine that runs on data. But the programming actually consists of putting things into the knowledge base.

PROLOG is a compiler that translates a high level description process into a low level, but very much faster, description of the same process. This can therefore be run directly on a computer and enables quite complex themes like medicine or law to be processed or thought about by computers.

We have seen how to present data in databases with our three models, but the biggest problem for expert systems is how to acquire data so that they can solve problems automatically, or semi-automatically. The computer must take human knowledge and human expertise and convert it into the symbolic data structures that we have discussed. To do this it must be able to 'learn', and this is probably the biggest stumbling block in artificial intelligence research today.

At present, the data for expert databases is accrued in a painstaking way. Individual computer scientists work with individual 'experts' slowly entering facts, assumptions and heuristics into computers. It has taken about ten years for a physician and a computer scientist to build INTERNIST, which diagnoses about 500 internal diseases and more than 3,500 manifestations of disease. With experience it should be possible to design and build expert databases in much shorter periods of time.

Glossary

architecture	the structure of the database; relational, hierarchical or network
database	a collection of stored data in one of various structures
database management system (DBMS)	software that handles all access to the database
data dictionary system	a filing system or thesaurus and index for the database
data independence	the immunity of applications to change in storage structure and access strategy
data model	the structure of the database as it appears to the user
data sublanguage	the subset of the host language concerned with retrieval and storage of information in the database
domain	a field, type occurrence or column of data in a relational database
encryption	a way of scrambling data
entity	an item or relationship in a database
hierarchical database	database structured like an inverted tree
inconsistency	one of two identical files has altered in some way so that they disagree
integrated database	contains data for more than one user and the data may be shared
key words	define documents in the data dictionary
meta data	data about the database, synonyms, for example
normalisation	in relational databases, it is the process of filling every row in every column with non-decomposable data
primary key	unique item in a stored record that identifies the record
privacy key	a group of characters added to data in the database to scramble it
program communication block	all access to hierarchical database
redundancy	unnecessary duplication of stored data
relational database	a collection of files normalised of different sizes
root segment	in hierarchical databases, the segment at the top of the hierarchy
segment occurrence	the smallest accessible unit in an hierarchical database
set type	an owner with or without members in a network database
sub-schema	the equivalent of the data sub-model in hierarchical databases
tuple	a record occurrence or row in a relational database

22

SOLID STATE ELECTRONICS

Light sensors

What is a light sensor?

As you might imagine, a light sensor responds to radiant energy in the form of visible light. However, for this to be of practical use, the sensor must *convert* this light energy into another, more useful, form of energy. For example: plants convert light energy into chemical energy; the eye converts light into electrical signals which are processed and interpreted by the brain; solar collectors, such as solar water heaters, convert light energy into heat; and solar cells convert light into electricity, which can either be utilised immediately or stored, in batteries.

Light energy can also be used to *control* rather than transform energy. For example, optoelectronic semiconductor devices are specially designed to make efficient use of the phenomenon that most transistors and diodes react to light to some degree, to control and regulate electrical energy. **Photoresistors**, **photodiodes**, **phototransistors** and **photothyristors** are examples of such devices.

We shall now go on to look at some of these components in more detail.

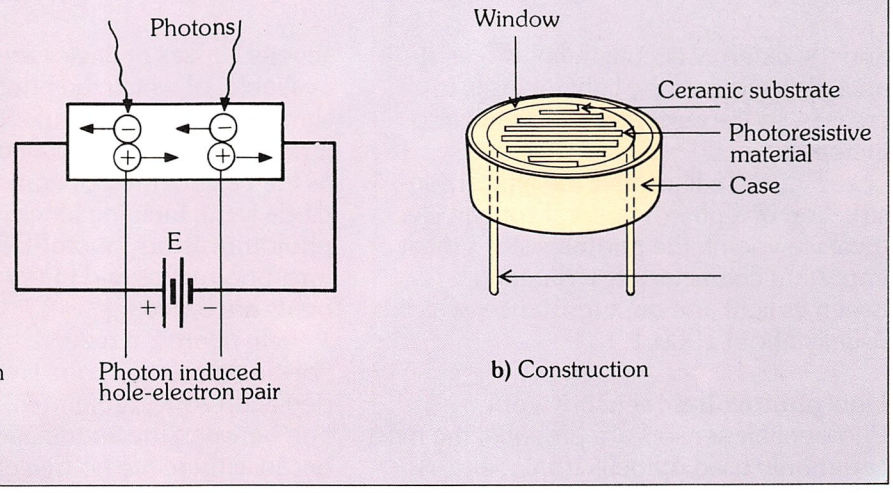
Table 1
Energy gaps for different materials

Material	Chemical symbol	eV at 300 K optical energy gap
Cadmium sulfide	CdS	2.4
Gallium phosphide	GaP	2.2
Cadmium selenide	CdSe	1.7
Gallium arsenide	GaAs	1.4
Silicon	Si	1.1
Germanium	Ge	0.7
Indium arsenide	InAs	0.43
Lead sulfide	PbS	0.37
Lead telluride	PbTe	0.29
Lead selenide	PbSe	0.26
Indium antimonide	InSb	0.23

Photoresistors

The action of a photoresistor is based on the principle that external energy provided to a material can remove electrons from their parent atoms creating free electrons and holes. This free electron then becomes available as a current carrier, thereby reducing the effective resistance of the material. The energy required to free an electron is known as the **energy gap** and is dependent on the type of material; *table 1*

1



1. A photoresistor. (a) operation; **(b)** construction.

lists the energy gaps for different materials.

There is a relationship between this energy and the wavelength of light that produced it which is given by the equation:

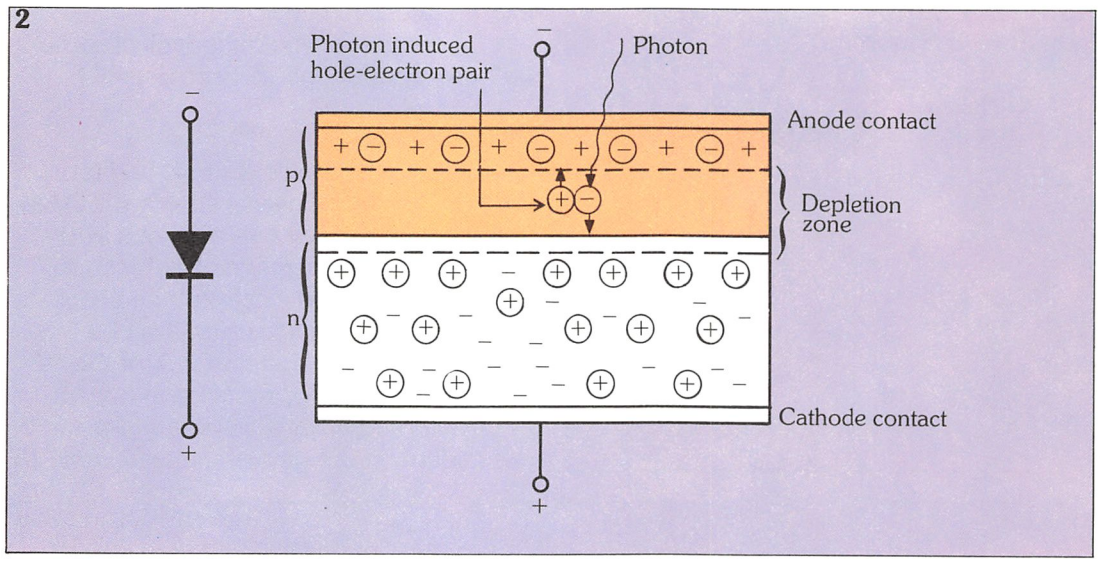
$$E = \frac{124{,}000}{\lambda}$$

where E is the energy in electronvolts (eV) and λ is the wavelength in nanometers (nm). Using this formula and values for E taken from *table 1*, we can see that the energy necessary to generate a free electron/hole pair corresponds to radiation wavelengths between about 500 and 6000 nm.

As *figure 1a* shows, a photoresistor does not need a p-n junction in order to operate: a layer of photoresistive material is connected to two leads. Because this sensor does not generate a voltage, an external voltage source must be used to make electrons flow through the sensor and the external circuit; the photoresistance decreases as the light intensity increases, so the current in the circuit also increases.

Figure 1b illustrates the typical construction of a photoresistor. From an electrical viewpoint, the photoresistor's most important characteristic is the ratio between its light and dark resistance, which is usually about 1000:1.

Above: Plumbicon television camera tubes for electronic news gathering and electronic field production. (Photo: Philips).

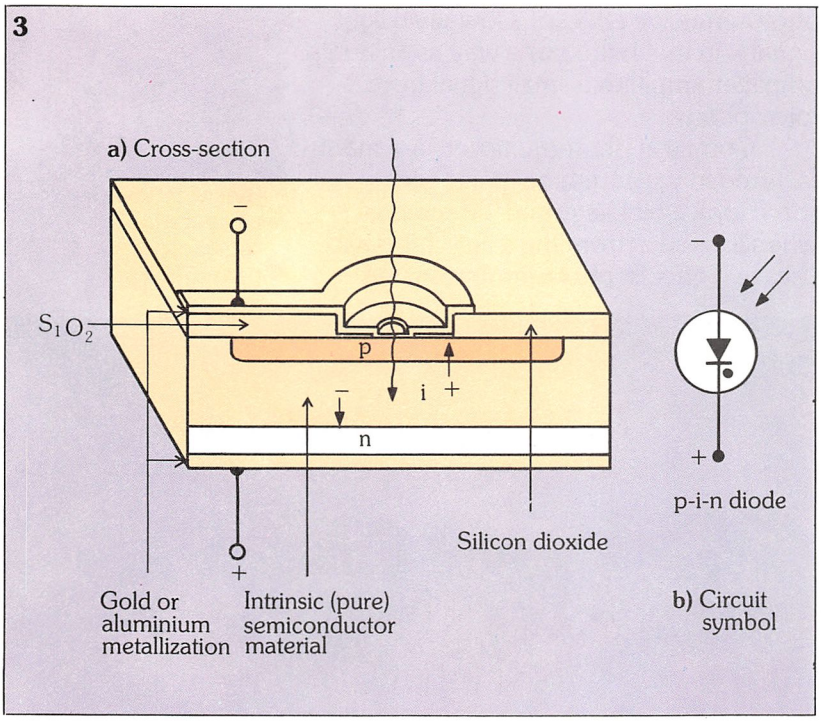


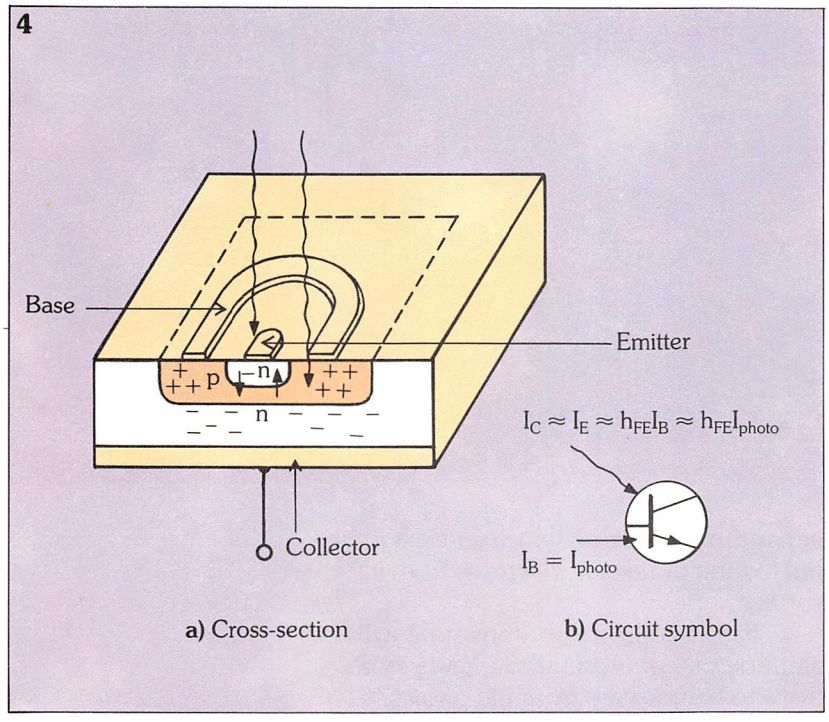
2. A reverse-biased p-n junction and depletion zone.

How photovoltaic sensors work

Photovoltaic sensors are probably the most commonly used optoelectronic sensors; several classes of device are commercially available, of which the photodiode is the simplest. These components use the p-n junction effects of semiconductor materials as the basis of their operation. The photodiode is the building block on which phototransistors, photoFETs, photothyristors, phototriacs and many other components are based.

In *figure 2* a reverse-biased p-n junction and depletion zone is shown. The depletion zone, remember, acts as insulation between the anode and cathode because there are no free electrons or





3. (a) Section through a p-i-n diode; (b) its circuit symbol.

4. (a) Cross-section through a photoresistor; **(b)** its circuit symbol.

holes present. When a photon with energy equal to that of the energy gap enters the depletion zone, it is absorbed. Hole/electron pairs are formed as a result of such absorption, but as there is an electric field present, these pairs are separated and move out of the depletion zone.

The electron (−) moves towards the cathode, while the hole (+) moves towards the anode. If the p-n junction is reverse-biased – like the one in *figure 2* – then current flows; on the other hand, if the hole/electron pair is formed *outside* the depletion zone, then it recombines (usually) and no current flows.

From this, you can see that if a p-n junction is to be used as a light sensor, then the p-region must be as thin as possible to reduce the possibility of recombination, and the depletion zone should be as wide as possible to improve sensitivity. A wide depletion zone has the added advantage that it reduces junction capacitance – this means that the photodiode will be able to respond to rapid changes in light levels.

The depletion zone can be widened in two different ways. A layer of intrinsic (pure) semiconductor material can be included between the n and p-type regions. This forms a **p-i-n diode** and a section through one of these devices is shown in *figure 3*.

The reverse-biased voltage applied to the device is positive on the n-material and negative on the p-material. Increasing this voltage provides the second method of widening the depletion zone and improving the device's photoelectric performance. However, since the leakage current increases with the rise in reverse voltage, the smallest possible reverse voltage should be used. These two conditions contradict one another so the final operating levels chosen are the result of a compromise between the two.

Phototransistors

In many applications, photodiodes are used in conjunction with an amplifier to increase the effect of the photocurrent. However, since a conventional transistor contains a reverse-biased p-n junction (the collector-base junction) and has the ability to amplify a current applied to its base, it possesses everything necessary for a photodiode and an amplifier – all in one package. *Figure 4* shows a cross-section of a typical phototransistor, and its circuit symbol.

When light falls on a phototransistor's collector-base junction, a base photocurrent flows. This current is multiplied by the phototransistor's forward current transfer ratio, h_{FE} (the ratio of the collector current

to the base current), to produce the **collector photocurrent**. Many phototransistors do not have an external base lead, but some applications use phototransistors with base leads to allow additional control over the device.

Solar cells

If the diode shown in *figure 2* is not biased, the continued exposure to light produces a voltage across it. When the anode is connected to the cathode via a load resistance, current flows through the circuit. This means that the p-n junction is capable of converting light energy directly into electrical energy. The p-n junction device that is designed specifically for this purpose is called a **solar cell**.

Since the purpose of a solar cell is to produce electrical power, it is designed to reduce power losses, capture as much radiant energy as possible and operate without external bias. These requirements result in a device that uses materials with low resistivity.

Solar cells also have a large surface area to capture as much radiant energy as possible. As the solar cell is not biased – to remove the electron/hole pairs from the depletion zone – the depletion zone has to be as thin as possible in order to minimise recombinations.

This combination of large surface area and thin depletion zone results in a large capacitance producing a slow response to changes in light levels. This makes the device unsuitable for use as a photosensor in most applications. We can see from this that while photodiodes and solar cells are both based on the properties of the p-n junction, they are, in fact, very different devices.

Solar cells are used in satellites and manned space flights to provide power for the on-board electronic systems. Solar cells are also used to power calculators, digital watches and even telephone exchanges. The typical conversion efficiency (ratio of power out to power in) of a solar cell is about 10% but research has now produced devices that are about 20% efficient.

Photo-emissive sensors

Photomultipliers are designed to multiply the very few electrons produced by the photo-emissive effect of a low-level light signal – in much the same way as an audio amplifier amplifies a small signal from a microphone.

Common photomultipliers are manufactured in vacuum tube assemblies – some look a bit like round TV screens when looked at from the side. This device uses two effects: **photo-emission** and **secondary emission** which cause a material to emit or release electrons from its surface.

Below: cathode ray tubes used in oscilloscopes. (Photo: Philips).

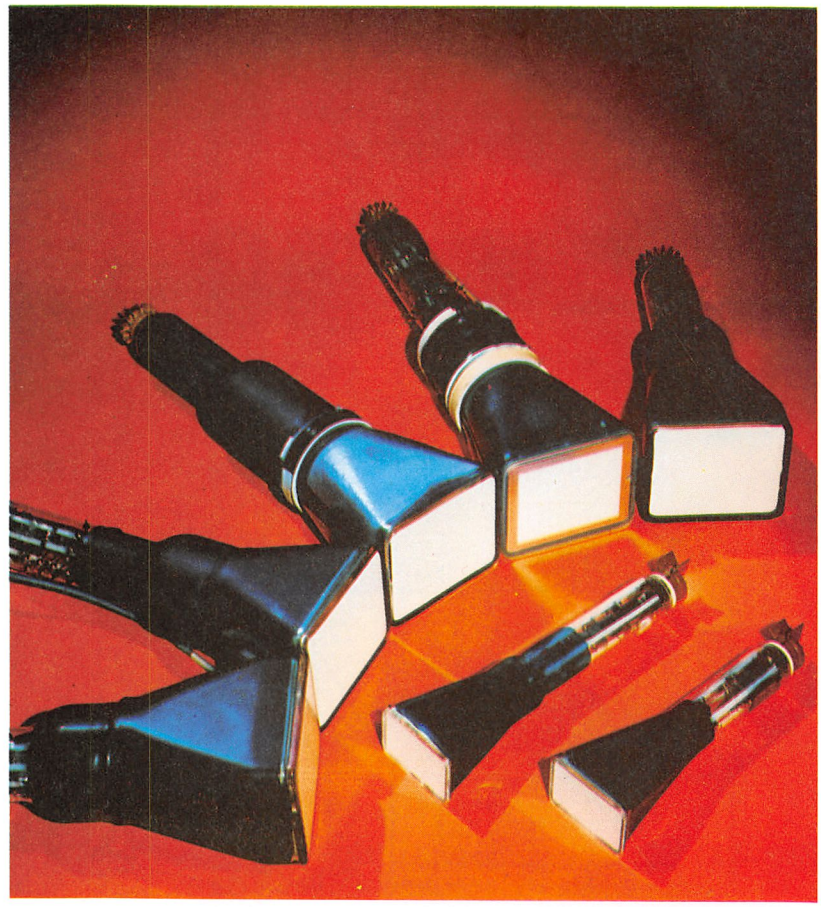
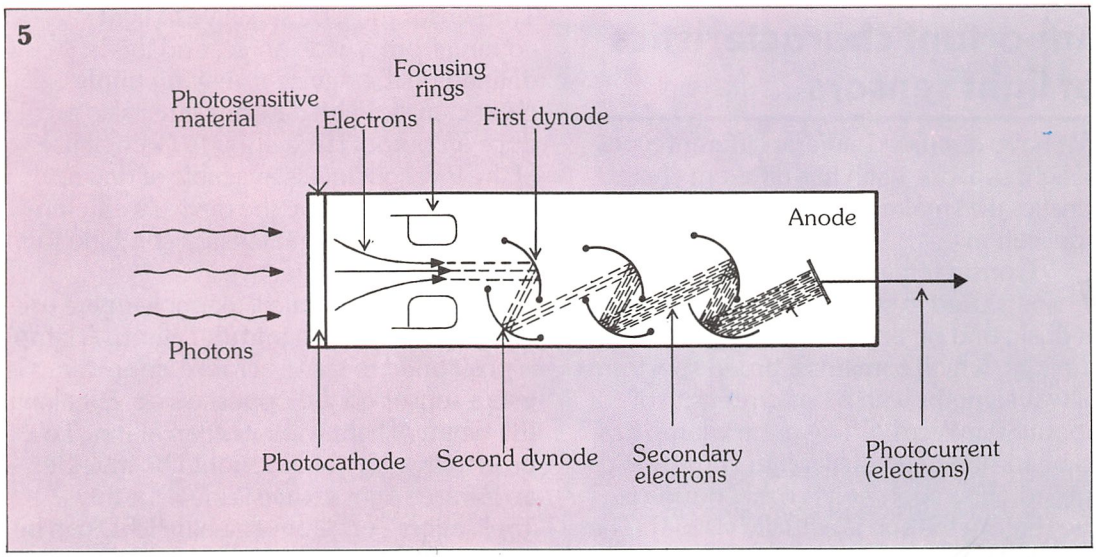
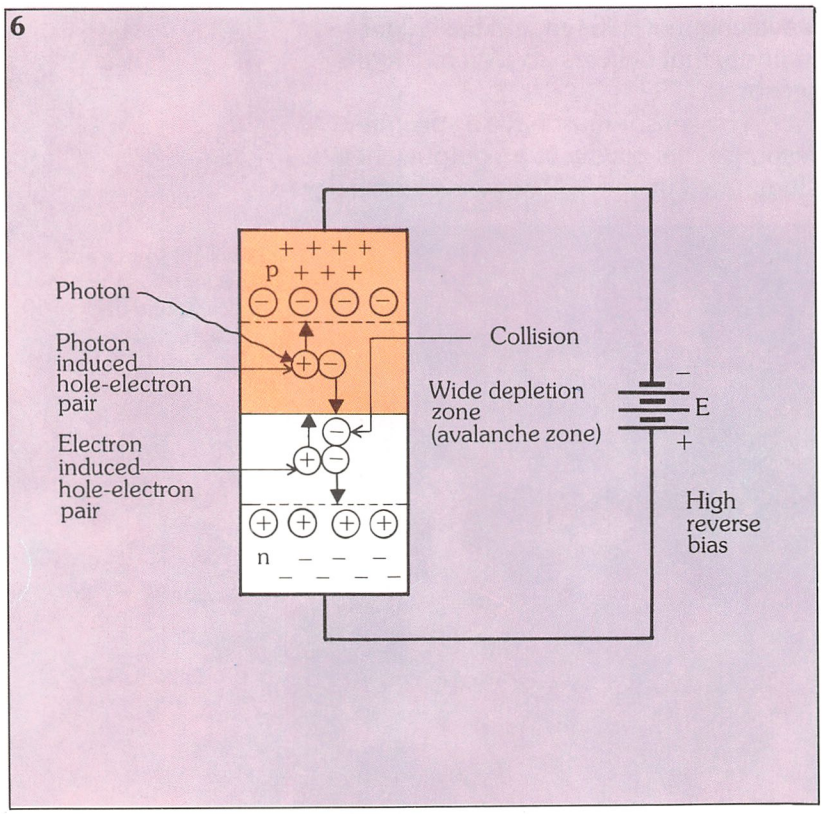


Figure 5 shows photons striking the cathode of a photomultiplier causing electrons to be released from the photocathode material through the process of photo-emission. These electrons are focused into a beam by an electrostatic field and are accelerated towards a curved plate or **dynode**. During this process, each electron gains enough energy to cause the release of two or more electrons from the dynode material – this process is called secondary emission. These secondary electrons are then accelerated towards

5. Action of a photomultiplier.





6. A photon with sufficient energy generates a hole/electron pair in the depletion zone leading to avalanche multiplication.

another dynode, where they cause more secondary emission, and this process is repeated several times.

The release of one electron from the photocathode (caused by the photon) may produce several thousand electrons as secondary emission is multiplied through several stages. The overall effect can produce current gains from 10^5 to 10^8.

The effect of light on a photo-emissive sensor may also be multiplied by the use of **phototransistors** or **photoFETs**. However, the semiconductor device that acts most like the vacuum tube photomultiplier is the **avalanche photodiode**. Avalanche breakdown, if you remember, refers to a condition in which a p-n junction is reverse-biased to the point at which electrons can be pulled from the atomic structure. A small amount of additional energy will dislodge electrons from their orbits, producing free electrons and holes. The reverse bias condition widens the depletion zone.

As *figure 6* shows, a photon with enough energy generates a hole/electron pair in the depletion zone. Because of the action of the high electric field, the hole with its positive charge moves up towards the negative side of the battery and the electron moves to the positive side. The electron moving in the high electric field is accelerated and collides with other bound electrons. Because of the high velocity when the electron collides, additional hole/electron pairs are generated which are also accelerated. These in turn can produce other hole/electron pairs, resulting in an avalanche multiplication process. One photon may produce up to 100 electrons in the avalanche photodiode.

In order to use these devices more efficiently as photosensors, an avalanche photodiode and a reference diode are combined inside the same package. This makes a temperature compensated circuit, and keeps the device gain constant, even with large changes in temperature.

Important characteristics of light sensors

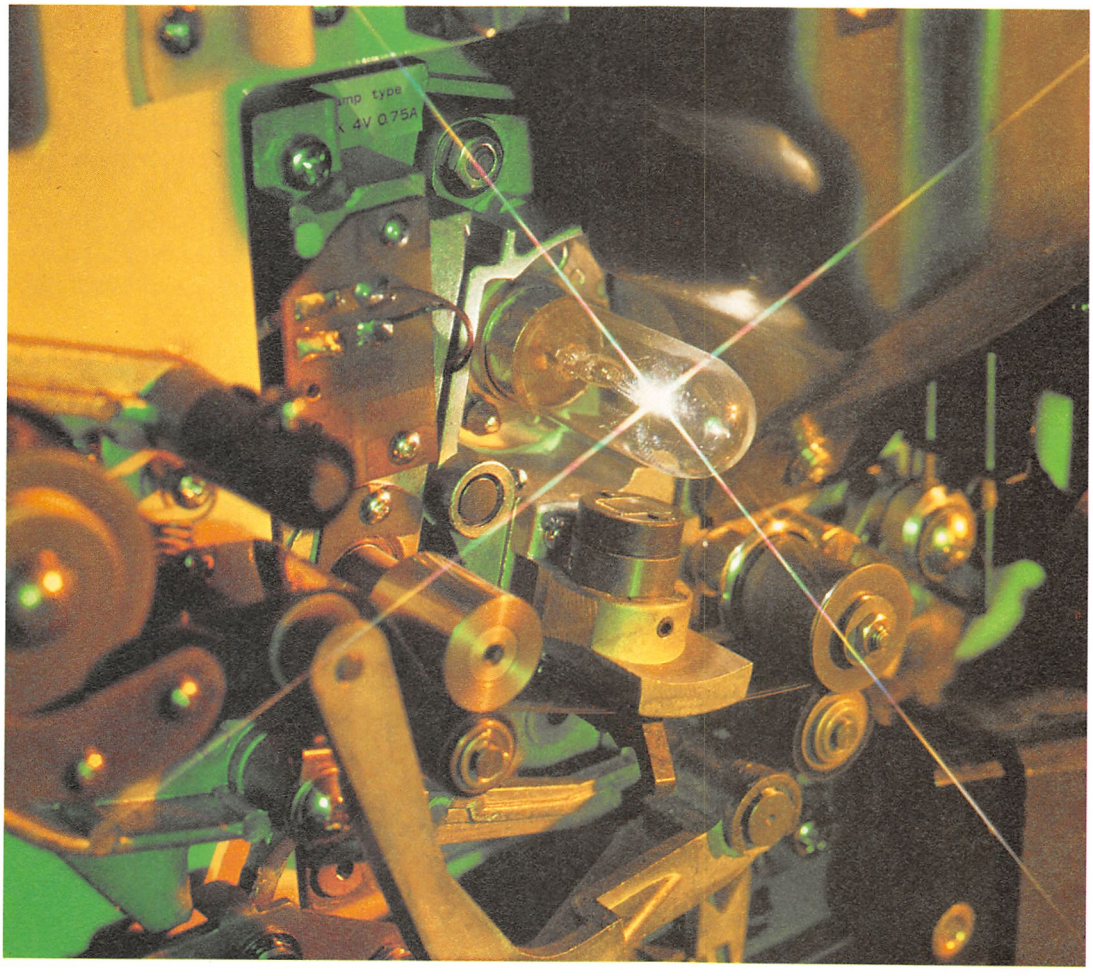
We have discussed several different types of light sensors: each has different characteristics that make it suitable for particular applications.

Consider the way in which a light sensor is used to switch an outside lamp on at dusk, and off at dawn. The light source is sunlight, which contains a broad spectrum of wavelengths with a peak intensity of about 80 $mWcm^{-2}$. The electric lamp has to be turned on at dusk (after sunset) and turned off as soon as it is light enough to see (before the sun is actually visible). The ambient light comes from a different direction at any given time. What characteristics must the light sensor have?

Well, of course it must respond to wavelengths found in sunlight. The range of wavelengths that affect a light sensor is defined by the device's spectral response. The sensor must be sensitive to light coming from a wide angle, and this is defined by the device's **viewing angle** characteristic. The detector must also produce an output large enough to be of use at the low light levels available at dawn: this is determined by the device's **efficiency**. All of these characteristics combine to make the system effective.

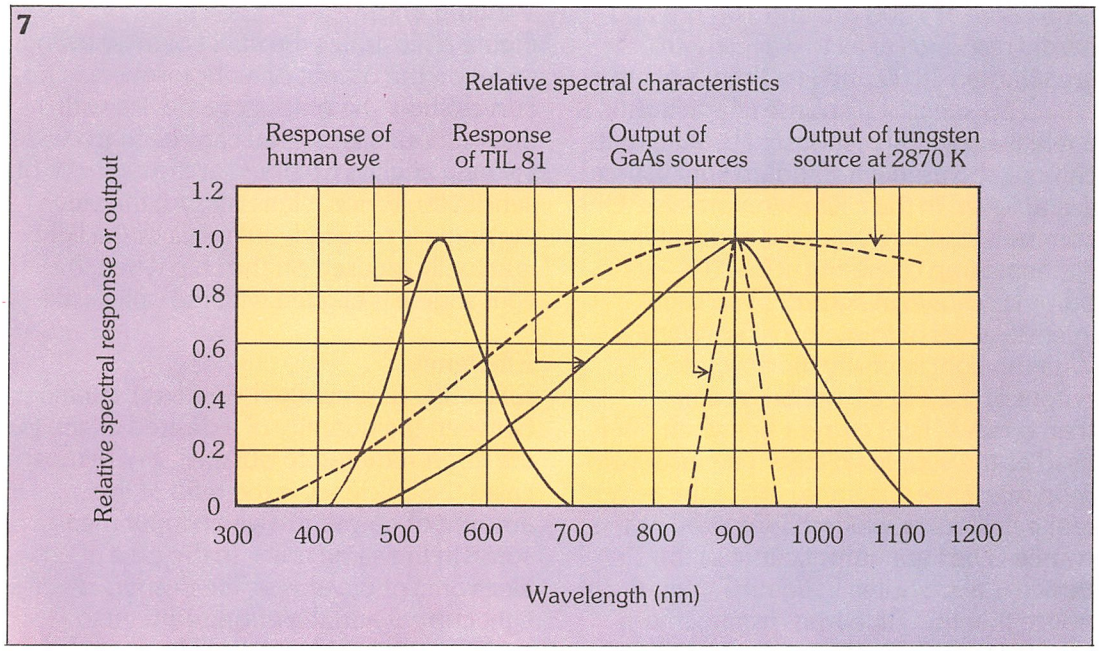
A second practical example of the use of light sensors is an intruder alarm. A lamp is positioned to shine across a doorway, into a sensor on the opposite side. Breaking the beam of light will set off an alarm. To be effective, the beam should be invisible – an infrared light source is ideal for this application. A gallium arsenide LED can be used which will emit infrared light at a wavelength of 930 nm and has a light intensity that delivers 25 μWcm^{-2} to the sensor.

The sensor must have a spectral response that produces an output when illuminated by the 930 nm wavelength. Its

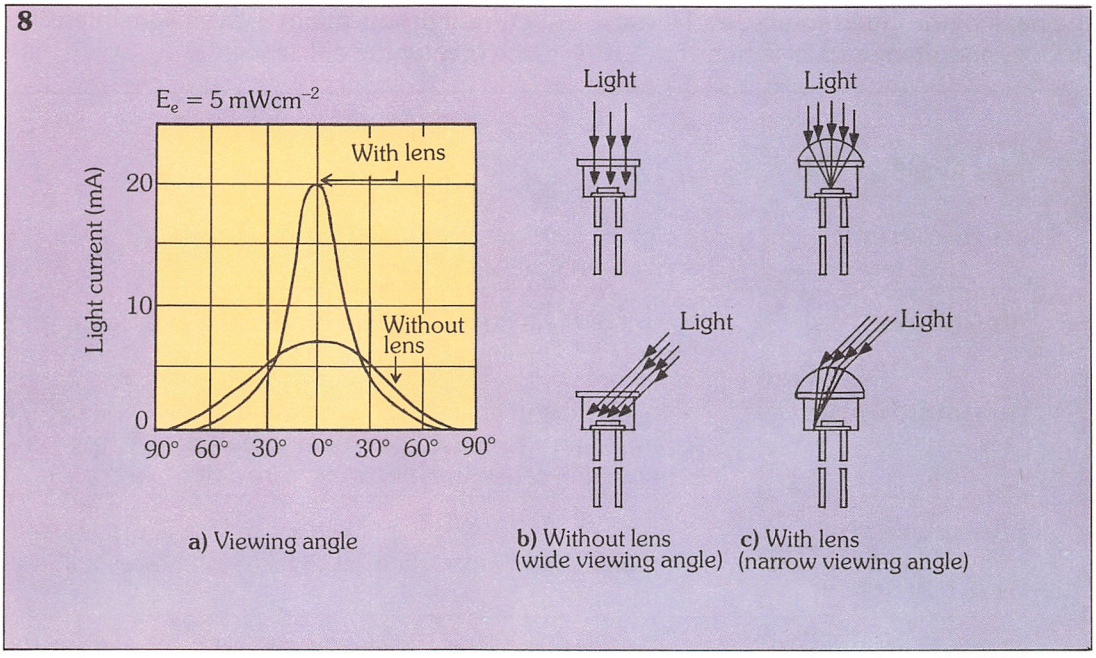


Neville Miles

Left: light source and detector for reading the optical sound track of 60 mm film.



7. Relative spectral characteristics of the TIL 81 and the human eye.



8. Effect of viewing angle on the output of a photosensor.

viewing angle must be such that it responds to light coming from the source, but is unaffected by light from a different direction. Finally, the efficiency of the device must be such that it will produce a change in its output with less than 25 μWcm^{-2} illumination.

Spectral response

The spectral response of a sensor indicates how the device responds to the same quantity of light at different wavelengths. It is important to understand that a sensor may sense light at one wavelength while it may not respond at all to that of another. As an example, the human eye can detect visible light, but cannot see radio waves.

Most light detector data sheets contain a graph similar to the one in *figure* 7 which compares the relative spectral characteristics of a phototransistor semiconductor detector (the TIL 81) and the human eye. The human eye responds to wavelengths from 400 to 700 nm. You'll notice that the peak response of the TIL 81 is just above 900 nm. It is only half as

responsive at 1000 nm and 700 nm and hardly responds at all to wavelengths greater than 1100 and less than 500 nm.

This spectral response characteristic is a relative spectral response; absolute units can only be used if a standard light source is employed in the calibration process. One common standard source is a tungsten filament lamp operating at 2870 K – its output is shown in *figure 7*. The usual intensity level of this lamp is 5 $mWcm^{-2}$. With this light intensity the absolute output of the TIL 81 will be greater than 5 mA of light current (I_L) when 5 V is used as the supply voltage. The most common reference is infrared, since the advent of the gallium arsenide (GaAs) 930 nm source. The light output curve of the device is also shown in *figure 7*. You'll notice that the TIL 81 has its maximum sensitivity at about 930 nm, which matches the peak of the GaAs source, and these devices are often used together.

Viewing angle

Figure 8 illustrates the effect of the viewing angle on the output of a photosensor. The curves show the output of a device with and without a lens. You can see that a wide viewing angle is obtained at the expense of sensitivity. When a lens is used, the viewing angle is reduced, but more of the light is actually focused on the chip when the light source is aligned with the optical axis.

Efficiency

Efficiency is normally defined as the ratio between the quantity of a desired effect, to the effort required to obtain it. So, in most cases the efficiency is the ratio of the amount of output to a given input measured in the same units. In the case of an electronic photosensor, the desired effect is light current and the effort required to produce it is light intensity. The more light current produced for a given light intensity, the greater the efficiency.

Glossary

photocurrent	electric current produced in a device by the effect of incident electromagnetic radiation
photodiode	a semiconductor diode that produces a significant photocurrent when exposed to light
photomultiplier	an electromultiplier that contains a photocathode. Light striking the photocathode causes electrons to be released (photo-emission) which in turn liberate more electrons from a dynode (secondary emission)
photoresistor	passive photosensitive device, manufactured from photoresistive material, the resistance of which decreases as the light intensity increases
phototransistor	a photosensor that consists of a bipolar transistor. When light falls on the collector base junction, a base photocurrent will flow controlling the transistor's action. Can be likened to an amplified photodiode
p-i-n photodiode	contains a layer of intrinsic (i-type) semiconductor between the p and n-regions. The depletion zone is thus contained within the intrinsic layer, improving sensitivity and spectral response
solar cell	device that uses the photovoltaic effect to convert the sun's radiation directly into electricity. Essentially a p-n junction with a large surface area and a high efficiency
spectral response	the characteristic of a photoelectric device that indicates its performance over a range of electromagnetic (light) wavelengths